2013

Analysis of Android Update Packages as a Method to Load Forensic Utilities and Malicious Applications to an Android Device

Mark Lohrum
Purdue University, marklbgsu@gmail.com

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Mark Lohrum

Entitled

Analysis of Android update packages as a method to load forensic utilities and malicious applications to an Android device

For the degree of _____

Is approved by the final examining committee:

Dr. Samuel Liles — Chair

Dr. Marcus Rogers

Prof. Raymond Hansen

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Dr. Samuel Liles

Approved by: Fatma Milli — Head of the Graduate Program — 04/12/2013 — Date

ANALYSIS OF ANDROID UPDATE PACKAGES AS A METHOD TO LOAD FORENSIC UTILITIES AND MALICIOUS APPLICATIONS TO AN ANDROID DEVICE

A Thesis

Submitted to the Faculty

of

Purdue University

by

Mark Lohrum

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2013

Purdue University

West Lafayette, Indiana

To my family, who encouraged me, even when I was several states away. And to my fiancé, Sophia, who encouraged me to continuously strive to be the best.

ACKNOWLEDGEMENTS

The author has many people to thank for assistance during this thesis. His advisor and committee chairman, Dr. Sam Liles, and Dr. Marc Rogers and Professor Raymond Hansen. Without this committee, this research would have been misguided and incomplete. John, for allowing use of hardware after working hours. Sophia, for being supportive and encouraging and always there. Eric and Matt, for listening to research ideas.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

GLOSSARY

Android – A Linux based operating system optimized for mobile devices, including smartphones and tablets (Daniel, 2009).

Android Device – A computing device, including but not limited to smartphones and tablets, which runs the Android operating system (Daniel, 2009).

Code Division Multiple Access (CDMA) – A type of digital cellular network, designed by Qualcomm, which spreads data across the entire available bandwidth (National Institute of Standards and Technology, 2007).

Electronic Evidence – "Information and data of investigative value that is stored on or transmitted by an electronic device" (National Institute of Standards and Technology, 2007).

Encryption – "Any procedure used in cryptography to convert plain text into cipher text to prevent anyone but the intended recipient from reading that data" (National Institute of Standards and Technology, 2007).

File System – "A software mechanism that defines the way that files are named, stored, organized, and accessed on logical volumes of partitioned memory" (National Institute of Standards and Technology, 2007).

Forensic Computing – The process of identifying, preserving, analyzing, and presenting digital evidence in a manner that is legally acceptable (McKemmish, 1999).

Forensic Copy – "An accurate bit-for-bit reproduction of the information contained on an electronic device or associated media, whose validity and integrity has been verified using an accepted algorithm" (National Institute of Standards and Technology, 2007).

Global System for Mobile Communications (GSM) – Cellular system designed in Europe which is used worldwide and employs the use of a SIM card (National Institute of Standards and Technology, 2007).

Hash – A mathematical function which converts digital information into a fixed length value. It should be computationally infeasible to derive the original information from the hash value. It should also be computationally infeasible to find any two digital messages with the same resulting hash values. Two example hash functions are MD5 and SHA256 (Thompson, 2005).

Personal Information Management (PIM) – "Data that contains personal information, such as: calendar entries, to-do lists, memos, reminders, etc" (National Institute of Standards and Technology, 2010).

RSA Algorithm – A cryptographic scheme developed by Rivest, Shamir and Adlemanwhich is the algorithm of choice for many online data communications, including financial transactions and secure electronic mail (Robinson, 2003).

Root – In Linux, root is a user with top privileges. To "root" a device or "gain root privileges" is to run a process as root. Root can also refer to the top level directory of a device. The directory "/" is the root directory. The directory /system is a "root level" directory because it exists at the root (Lessard & Kessler, 2010).

Signature – "Digital signatures provide authenticity and nonrepudiation of electronic legal documents" (Boneh, 1999).

Smartphone – "A full-featured mobile phone that provides users with personal computer like functionality by incorporating PIM applications, enhanced Internet connectivity and email operating over and Operating System supported by accelerating processing and larger storage capacity compared with present cellular phones" (National Institute of Standards and Technology, 2010).

ABSTRACT

Lohrum, Mark A. M.S., Purdue University, May 2013. Analysis of Android Update Packages as a Method to Load Forensic Utilities and Malicious Applications to an Android Device. Major Professor: Dr. Samuel Liles.

Android devices are extremely popular and are projected to stay popular. Both forensic tools and malware exist designed specifically for Android devices. The purpose of this study is to explore a new method of loading forensic tools and malware to Android devices. This new method is the update module, which is used to install updates to the operating system. This thesis proposed and completed research to test four different custom update packages on three different Android devices. Two of the update packages contain forensic utilities and the other two contain malicious tools. An update package collecting web history using the device's API successfully executed on all three devices, and the other three update packages only successfully executed on the oldest device, a Nexus One. The cause for the differing results from the Nexus One to the other two devices is the version of the operating system on the device. The Nexus One ran the oldest version of Android which did not contain the same security measures as the newer operating systems on the other two devices. The newer devices had these security measures which stopped the update packages from executing successfully. The researcher concluded that these update packages are not ready to be used currently.

CHAPTER 1. INTRODUCTION

1.1 Introduction

The Android operating system, designed for smartphones and other mobile devices, is the most popular mobile operating system in the United States (Eadicicco, 2012). This operating system now accounts for half of current American smartphones in use (Weiss, 2012) and continues to see growth. The Android platform is also a popular platform for developers, as over 600,000 apps are in the "Play Store", or the market for downloading apps (Fingas, 2012).

As long as consumers, including criminals, use these devices, law enforcement must be capable of extracting data of a forensic interest from these smartphones. This personal data, such as contacts, text messages, web history, and call logs, can be extracted using commercial forensic tools. Many of these tools are evaluated in the book *Android Forensics: Investigation, Analysis, and Mobile Security for Google Android* (Hoog, 2011). The book states that the only requirement for these tools is that USB debugging, a setting allowing for apps to be installed via a Universal Serial Bus (USB) connection, is enabled. If the phone is screen locked, the investigator does not know the pass code, and USB debugging is not enabled, then USB debugging cannot be enabled and these tools cannot be used. Hoog's book details some methods of circumventing the pass code, though it acknowledges that pass code bypasses are not consistently reliable.

"While there is no guaranteed method, there are a number of techniques have have worked in certain situations" (Hoog, 2011 p. 203).

With the popularity of the Android device on the rise (Weiss, 2012), mobile malware aimed at this operating system is also becoming more common. Mobile malware attacks are aimed largely at Android devices, as around three quarters of all examples of mobile malware are designed for the Android operating system (Greenberg, 2012).

These concerns present a need for security experts to understand these devices. This study presents a previously unexplored method of attacking Android devices. This attack can be used for benevolent purposes, such as extracting data from the device to assist law enforcement, or malevolent purposes, such as installing malware.
This chapter introduces the topic presented in this study, along with the scope, significance, and boundaries of the research to be performed

1.2 Statement of the Problem

Android devices, including smartphones and tablets, contain a recovery mode for updating the operating system and firmware. This module can be exploited for other purposes. These purposes can be beneficial or they can be malicious. For this research, update packages which are considered beneficial are those which the law enforcement community could potentially use for extracting data from a suspect's Android device in a forensically sound fashion. Update packages which are considered malicious are those which can carry out malicious tasks, including planting evidence on a device or adding malware. Custom update packages can be written for these purposes, creating a vulnerability that has not been explored. This study answers the following question: can

update packages be used to deliver malicious payloads and/or utilities beneficial for a forensic investigation on current Android devices?

1.3 Significance of the Problem

As discussed in Chapter 2, most commercial Android forensic tools operate on a live device and depend on the user having the ability to install an application on the device. If the device is locked and the password is unknown, and USB debug mode is disabled, forensic tools are rendered useless on Android devices. The forensic aspects of this research demonstrate a method to circumvent this shortcoming of Android forensic tools. The criminal aspects of this research demonstrate abilities to attack a device in novel, potentially frightening ways. These innovations make this study significant.

Android devices are very popular; over half of smartphones users in America now use Android phones (Weiss, 2012). The International Data Corporation projects Android to remain the top mobile operating system through at least 2016 (International Data Corporation, 2012). With this many phones running Android being sold, this study is significant.

Like other smartphones and tablets, Android devices contain a wealth of information about the user (Hoog, 2011). Besides containing the normal phone information, including contact lists, phone calls, and text messages, an Android smartphone contains calendars of the user's activities, web browsing activity, applications, multimedia files, and more (Daniel, 2009). This amount of evidence readily available in Android devices makes this study significant. Android tablets contain a great amount of personal information, including calendar entries, web history, e-mail, office documents, and more.

1.4 Purpose of the Study

This study is aimed at documenting a previously unexplored attack vector present in nearly all Android devices. As stated in Chapter 2, nearly all Android devices have a recovery mode. This study is aimed at both documenting this attack vector and determining the effectiveness of procedures, both benevolent and malevolent, on the devices. This is not an endorsement of a forensic procedure or a study of admissibility of a forensic procedure. The goal is to create a study useful for academics, security experts, and Android enthusiasts while creating awareness of the possibilities of attacking an Android phone through its recovery mode.

1.5 Assumptions

This study contains a few assumptions. It is assumed that the Android devices used are in good working order. This includes the "factory reset" functionality, which is used to set the devices to a known state. It is assumed that a "factory reset" is a reliable way to return a device to a known state. Since a version of the Android operating system was built and installed on each smartphone and tablet, it is assumed that the download of the source code, compilation of the operating system, and uploading of the operating system to the device all goes smoothly. It is assumed that the all software tools used in this research work properly. Three different devices which each ran a different version of the Android operating system were used. It is assumed that each device well represents other Android devices which run the same version of the operating system. Also, it is assumed that the variety of the devices chosen allowed for the results of this study to be applicable to Android devices in general, as opposed to only the devices selected for this study. As discussed in Chapter 2, other Android related studies did not include the variety of

devices, in terms of manufacturer and operating system version, that were chosen for this study.

1.6 Delimitations

This study is a proof of concept study. This is not intended to be an analysis of a tool which is ready to be used by a broad community. Given this delimitation, a formal admissibility study is not discussed, but the sizes of the update packages and their impacts on the device are discussed.

This study is neither aimed at identifying the locations of interest of files on Android devices nor producing new forensic tools or malware. New attacks are revealed in this study, but this is not intended to be used as a guide for securing a handheld device; instead, this is meant to demonstrate some previously unexplored capabilities of a specific exploit of the device.

Three different devices were chosen for this study, each of which run a different version of Android. This is intended to give a representative sample of Android devices.

1.7 Limitations

As stated in Chapter 3, the researcher modeled an original equipment manufacturer (OEM) when completing this study. Only a certain set of devices were analyzed. The justification for these specific devices is in Chapter 2. A limitation of this research is the devices chosen. The devices do represent three different manufacturers, three different versions of the Android operating system, and two different form factors, but only three devices were picked. These three were all freely available to the researcher. Finances limited the device selection, so only three devices were used in this study. Due to constraints of availability, the smartphones available are all GSM. No CDMA Android

smartphones are available to the researcher which satisfy the requirements. These device limitations do not substantially affect the results of this research as three different manufacturers and operating system versions are represented in these devices. While a great wealth of goals, both forensic or malevolent in nature, can be achieved through creating custom update packages, only a small set of focused tasks is represented. One of the tests in this study involves creating a bitstream image of the device. When creating a bitstream image of a hard drive, the image is traditionally deemed authentic by calculating a hash of both the original hard drive and the image and compare for accuracy. The device which is image cannot be hashed in this manner as the device is powered on and constantly changing. This presents a limitation in determining accuracy of the created bitstream image. Other methods of determining accuracy, including verifying the size and specific contents of the device, are used instead.

CHAPTER 2. REVIEW OF THE LITERATURE

The Android platform has gained a great amount of popularity since its initial release in 2008. With that popularity comes an interest in forensic extractions of data from the devices. Also, an interest in the development and the spread of malware arises. The method used in this research requires technical knowledge regarding the Android recovery mode and creating custom update packages; this technical knowledge is found online in technically-oriented websites. The method for this study is similar to methods used in other Android forensics papers. In many ways, this method is more thorough than the methods used in these other papers. The devices chosen for this study support the methodology.

2.1 Android

Android is not a phone; it is an operating system. In 2005, Google acquired Android Inc, which at the time was a 22 month old company based in California that had been operating "under a cloak of secrecy" (Elgin, 2005). Common speculation was that Google had interest in the hardware of phones, but their focus was to instead develop an Android operating system for mobile phones and release it for free (Daniel, 2009). On September 23, 2008, T-Mobile announced the forthcoming release of the G1. Billed as the world's first Android powered phone, it was manufactured by HTC (HTC Corporation). The G1 sold around one million units in the United States, leading to the

release of the G2, the successor to the G1; the G2 sold around one million units in its first month (Daniel, 2009).

Mobile phone manufacturers would either spend a great amount of time and money developing their own operating systems or purchase one for a hefty price. With the availability of the Android operating system, manufacturers can have a full operating system ready for their phones at a lower cost. Manufacturers can either launch Android on phone hardware as is or they may modify and customize it for their own needs (Daniel, 2009), such as with the Motorola Droid or the Nexus One, sold by Google and manufactured by HTC.

Data is stored on the phone in Android powered phones in a simple file structure, and most data about the user is stored in a root level folder called /data. While many other root level folders exist, this is the root directory of most importance to a forensic investigator (Mohindra, 2008).

The file system initially utilized by the Android operating system was YAFFS2, or Yet Another Flash File System 2 (Hoog, 2009). This is an open source file system that is found on some flash devices. Data is retained on the device longer than on other, more traditional file systems. Most computers cannot natively read a YAFFS2 image, but the unyaffs2 utility, available on Google Code, can be used to extract data from a YAFFS2 image (Penguin.lin, 2012). Newer phones have been using EXT4 as a file system instead. This file system can be read natively in Linux and can be read with most computer forensic tools, such as FTK Imager (Hoog, 2011).

2010 sales for Android increased worldwide over 800% from the previous year; Android's market share of operating systems jumped from 3.9% to 22.7% (Wilcox, 2011).

During the year, sales of phones running Android surpassed sales of iPhones (Burnette, 2010). In July 2012, Android passed a major milestone when a report showed that just over 50% of U.S. smartphone owners use Android devices (Weiss, 2012).

2.1.1 Android Forensics

There are companies which develop mobile forensic products which support Android powered devices. These products include Oxygen Forensic Suite (Oxygen Software Company, 2011), Paraben Device Seizure (Paraben Corporation, 2011), .XRY by Micro Systemation AB (Micro Systemation AB, 2011), Universal Forensic Extraction Device (UFED) by Cellebrite (Cellebrite Mobile Synchronization LTD, 2011), and AFLogical by viaForensics (ViaForensics, 2011).

There are two papers regarding testing of Android tools aimed at comparing these tools to each other and the completeness of the results they obtain. One work, appearing in the book *Android Forensics: Investigation, Analysis, and Mobile Security for Google Android*, tests several different tools on a Motorola Droid. It was noted that only one phone was used, and neither the phone nor the SD Card were "cleaned" between testing (Hoog, 2011). Another study utilized two different phones, both by Motorola, and tested three different tools (Kovacik & O'Day, 2010). This paper proposed a methodology regarding investigating Android phones quickly and inexpensively; however, the authors did not mention that in order to create a complete backup of the phone to examine the storage at a bitstream level, an alternative recovery program, such as ClockworkMod, must be installed.

Mobile phones can be analyzed at a bitstream level by physically taking the device apart and analyzing the chips which store data; this method is not limited solely to

Android devices (Mooij, 2010). This process is considered to be quite effective, but it is expensive and requires an examiner with the proper skills (Hoog, 2011).

There is not a great amount of academic literature regarding Android forensics, but some documents do exist. One paper explores the issue of rooting a phone, then analyzes three different methods of forensic acquisitions of potential data: physical imaging (which requires rooting the phone), logical acquisition of certain files (which requires rooting the phone), and using a Cellebrite kit to acquire data from the phone (Lessard & Kessler, 2010). Another article deals with anti-forensics, as opposed to forensics. This discusses methods of clearing out evidence effectively and efficiently using a tool called AFDroid (Distefano, Me, & Gianluigi, 2010). Mohindra (2008) discusses both the Android architecture and Android forensics. The discussion of evidence acquisition utilizes the Android Debug Bridge, which requires the device to be set to allow USB debugging. Removing key user files cannot be done using Android Debug Bridge from an actual device as permissions forbid this; however, this acquisition can be completed from a virtual machine device, which is a tool intended for Android developers, as done in this paper.

As mentioned, there is an Android forensics book available with much great information regarding the operating system, acquisition of data, and investigations (Hoog, 2011). This book covers some Android forensic tools, along with other methods, including potential methods to bypass a screen lock, extracting data from the Android Debug Bridge, and imaging the included SD card.

2.1.2 Android Malware

Android malware is a concern for security experts. A 2012 study concluded that around 75% of malware targeted at mobile devices is designed for the Android operating system (Greenberg, 2012). Security experts recently discovered legitimate-appearing applications which download malware to the mobile device, and these applications could have been downloaded to up to 100,000 devices (Naraine, 2012). F-Secure Labs, a security firm, discovered 51,447 pieces of malware for Android in third quarter 2012 (F-Secure Labs, 2012).

A 2013 report by Commtouch, an Internet security company, detailed the number of unique instances of Android malware discovered, and these numbers were significantly higher than those of F-Secure Labs. In January 2013 alone, over 178,000 unique malware samples for Android were discovered (Commtouch, 2013). This was actually lower than the December 2012 findings of over 214,000, which, according to the report, "may have been an anomaly." With the exception of December 2012, the number of unique pieces of malware discovered by Commtouch rose every month from July 2012 to January 2013. Given these numbers, Android malware, its spread, and mitigations must be a focus of research for mobile security experts.

2.2 Background of the Methodology

The following sections provide background information for the methodology used in this study. The researcher was unable to find any published academic papers regarding the technical aspects of this research, so the information in this section is derived from technical sources on the Internet.

2.2.1 Android Recovery Mode

According to Android developer documentation, the recovery system is “the separate partition that can be used to install system updates, wipe user data, etc” (Android, 2012). The researcher has personally used and explored many Android devices, perhaps as many as 50, and the only of these which did not have a recovery mode were from untrustworthy manufacturers, shipped from China, and sold inexpensively on eBay. All of the devices the researcher has used which were manufactured by reputable manufacturers, including Motorola, HTC, Samsung, LG, and Asus, contained recovery modes.

Recovery mode is accessed after the phone has been powered off. A combination of buttons is used to power on the device into recovery mode, and this button combination is different phone device to device. To boot into Recovery Mode on the Galaxy Nexus, the user needs to power the device off, then hold both volume buttons and the power button, and then choose from an on screen menu to boot into recovery mode (Verizon, n.d.).

The default recovery mode for Android devices includes four options: reboot the system, apply an update, wipe data / factory reset, and wipe the cache partition (Verizon, n.d.). Any update package must be digitally signed to be installed in the Android recovery mode, and the signature is verified before installation (Android, 2012). The wipe data / factory reset functionality is defined in Android documentation as “reboots the device and wipes the user data partition. This is sometimes called a 'factory reset', which is something of a misnomer because the system partition is not restored to its factory state” (Android, 2012). In this study, the phrases “factory reset” or “reset to a

factory state" refer to this functionality. As stated in Chapter 1, an assumption of this research is that a "factory reset" is a reliable way to return a device to a known state.

2.2.2 Alternate Recovery Mode

An alternate recovery mode, such as ClockworkMod, can be installed on many Android devices (ClockworkMod, n.d.). This has additional features beyond the standard recovery mode, including the ability to bypass signature verification when installing update packages and Android Debug Bridge connectivity, which allows access to a shell with root privileges (Xda Developers, n.d.). The Android recovery mode is installed on devices by default, so the user must add ClockworkMod or an equivalent recovery mode to access its extra functionality (ClockworkMod, n.d.). This study does not use alternative recovery modes. Instead, this study explores an exploit in the default recovery mode. If a device happens to have ClockworkMod installed, the device can simply be connected to a computer and accessed via a root shell using the Android Debug Bridge. With this connectivity, all of the objectives in this research can be performed. One should not assume that a device examined for an investigation has an alternative recovery mode; one must assume that the default recovery mode is on the device.

2.2.3 Android Updates

Official system updates are distributed to Android as zip files which are digitally signed to prove authenticity (Android, 2012). These zip files are distributed through the cellular network, or they can be downloaded to the device and installed manually using the recovery mode (Imran, 2012).

2.2.3.1 Writing Custom Updates

Developers can write custom update packages. These update packages can be found on the Internet to root a device (IndefactorX, 2009) or to install a custom recovery mode (ClockworkMod, n.d.). These are not the only two possible uses for a custom update package, but there are many such downloads on the Internet.

The structure of update packages is consistent from one update zip file to the next. At the root of the zip file is a directory META-INF. Within are three files, CERT.RSA, CERT.SF, and MANIFEST.MF, which are created during the package signing process and contain both the signature created from the private key of the signer and a list of all contained files with a hash of each file (Yang, 2012). Within this directory is a directory structure of com/google/android, and within the android folder are two files, update-binary and updater-script. The file update-binary is a binary which need not be changed from one update package to the next. The script updater-script follows a distinct scripting language (Londatiga, 2010). This script can be edited using any text editor and will change from one update package to the next. Also at the root directory, alongside the META-INF directory, can be other files used in the update, such as files to be installed on the phone or even entire directories (McGhee, 2010).

2.2.3.2 Signing Update Packages

As stated above, updates must be digitally signed to be installed in the Android recovery mode, and the signing process adds three files to the update package. These are signed using a Java jar file called signapk.jar, which can be used to digitally sign Android update packages or Android apk files for instaling applications (Atomicdryad, 2010).

These update packages are RSA signed with a private key, which the original equipment manufacturer (OEM) is responsible for securing (Android, n.d.). The matching public key is located on the device at /system/etc/security/otacerts.zip; if the signature of the update package matches the public key on the device, the update package is verified and installed (Android, 2012.).

2.2.4 General Method

The general method for this study revolves around comparing a device in a known state to the device in a new state after performing a specific task. For each of the devices used in this study and for each update package tested, the device began in a known state. To create a known state for the device, the device had been reset to a factory state using built in functionality and was next populated with a specific set of data. This data included web browsing and installing apps. The device was then booted into recovery mode and executed an update with one of the update packages created for this study. The device was then rebooted and the new state of the device after executing the update package was compared to the known state of the device. This method was repeated multiple times for each device and for each update package.

This method is similar to a chapter in Hoog's book *Android Forensics: Investigation, Analysis, and Mobile Security for Google Android*. This chapter includes a study of several mobile forensics tools for Android. The device, a Motorola Droid, was populated to a known state. Several mobile forensics tools were tested on the device. The results produced by those forensics tools were analyzed and compared to the Droid in the known state.

There are three key differences between the study in Hoog's book and this research. This research used three different devices by three different manufacturers, but the study in the book used one device. This research repeated each test on each device five times, but the study in the book only ran each test once on the device. Also, the method for this research included completing a factory reset before each repetition and populating the device with a specific set of data, where the study in the book's method involved populating the device once and then running one test after another on the device. This third difference allowed the researcher a better view of what is on the device in this research. The devices were populated with an exact known set. In the study in the book, the researcher could not account for changes made by a forensic tool. Each forensic tool would analyze the device as it existed after the previous tool was run on the device, so the device was no longer in a known state.

Another paper (Kovacik & O'Day, 2010) also analyzed computer forensic tools and contained the same issue as the previous paper of not starting the device before a test at a completely known state. Three computer forensic tools were run on two different devices, both by Motorola. Besides the three forensics tools, the researchers also performed “manual forensics,” which involves obtaining a bitstream image of the device and using traditional forensic tools, including file carving and a hex editor, to perform an examination.

The Kovacik & O'Day method did not involve any repetition, unlike this research. Also, this method does not begin with each device in a completely known state. The paper states “forensic artifacts within the phone were initially counted and each methodology was compared based on what artifacts it could find.” This implies that the

researchers did not have complete control over the contents of the device. Also, the paper states that the researchers "had no way to ensure that no data was altered or changed on each device as various methodologies were utilized, as each methodology typically involved changing data on the phones in some way, shape, or form. Thus it is possible that the quantity of a certain forensic artifact changed during the research process." This methodology in this research includes resetting each device to a completely known state before running a test, so the researcher knew the state of the device without question before running each test. Also of note in the Kovacik & O'Day paper is the manual forensics method, which included obtaining a bitstream image of the device. This requires either rooting the device or installing a custom recovery mode, such as ClockworkMod; the researchers did not specify exactly how they obtained the bitstream image.

A third Android paper (Distefano, Me, & Gianluigi, 2010) used a similar process to this research. This paper regards anti-forensics techniques for Android devices. A single Samsung device was used to run experiments to determine the effectiveness of anti-forensics tools. These tools were designed to remove evidence from a device. Two different experiments were run. The first experiment, testing the evidence export process, is as follows:

- Obtain a bitstream image of the device
- Execute the anti-forensics tool being tested
- Run a commercial mobile forensics tool on the device
- Obtain a second bitstream image of the device

The second experiment process, testing the evidence destruction process, is as follows:

- Obtain a bitstream image of the device
- Execute the anti-forensics tool being tested
- Obtain a second bitstream image of the device
- Uninstall the anti-forensics tool
- Run a commercial mobile forensics tool on the device
- Obtain a third bitstream image of the device

These methods created a bitstream image of the device before testing. A set of tests was performed on the device, then another bitstream image was created. This process allowed the second bitstream image to be compared to the first so changes to the device could be identified. This process is similar to the method used in this research. Instead of obtaining a bitstream image of the device and using this image as a known state, the devices were populated to a known state. Instead of creating a second bitstream image of the device to compare to the first bitstream image of the device, certain observations, specified in Chapter 3, were noted for each test. Both methods involve comparing the results of a test on a device to a known state of the device before the test. The process of obtaining a bitstream image in the Distefano, Me, & Gianluigi paper required installing an alternative recovery mode on the device. This was not performed in this research since the Android recovery mode was used in this method.

2.2.5 Devices

The methodology for this research allowed the researcher to act as an OEM. The OEM maintains the private key used to sign updates which the device can install. Should this private key be compromised, anybody with the required technical knowledge can build update packages similar to what are tested in this study. The researcher designed a

methodology allowing for this private key to sign update packages, so the builds of Android used must include the matching public key. The way the researcher is going about this is to build Android from source code. Google supports these builds for some devices, so the devices selected for this study are from these supported devices.

The Android source code is open, and instructions to build it are online (Android, n.d., Building for devices). As seen on the referenced website, only a specific set of devices are supported. Drivers for these devices are available online (Google, 2012), and these must be included in any Android builds. Ten such devices are supported as follows (Google, 2012):

- Nexus 10
- Nexus 7 (Wi-Fi)
- Galaxy Nexus (GSM/HSPA+)
- Galaxy Nexus (Verizon)
- Nexus S
- Nexus S 4G
- Motorola Xoom (US Wi-Fi)
- Nexus One
- Motorola Xoom (Verizon)
- Galaxy Nexus (Sprint)

Three of these devices were chosen to represent three different manufacturers, three different versions of Android, and two different form factors. The first device is the Nexus One. Gingerbread, or Android 2.3, was built and installed on this device. This smartphone was released in January 2010 and was manufactured by HTC. The second

device is a Nexus S. Ice Cream Sandwich, or Android 4.0, was built and installed on this device. This smartphone was released in late 2010 by Samsung. The final device, a tablet, is the Xoom. Jelly Bean, or Android 4.1, was built and installed on this device. The Xoom was released in 2011 by Motorola.

Building Android for these three devices allowed the researcher to act as an OEM and have access to the private key used to sign software updates. This private key was used to sign the custom update packages.

CHAPTER 3. METHODOLOGY

The methodology for research began with building the Android operating system for each device. To reflect the nature of the power which original equipment manufacturers (OEMs) have in possessing private keys for these devices, the researcher imitated an OEM by building from source the operating system for each device. Specific tasks, both of a benevolent and malevolent nature, were executed for each device. Details of this methodology are discussed in this chapter.

3.1 Research Devices

Chapter 2 details the research devices and the justification for these three devices. The three devices used were an HTC Nexus One running Android 2.3, a Samsung Nexus S running Android 4.0, and a Motorola Xoom running Android 4.1. The following photograph contains all three devices. From left to right, the Nexus One, Nexus S, Xoom.

Figure 3.1 Photograph of the Three Research Devices

3.2 Research Computer

The Android operating system must be downloaded and built for each device. A 64-bit workstation running Ubuntu 10.04 was used. Long before this research was proposed, this workstation was prepared for Android builds, which required installing a lot of prerequisites (Android, n.d., Initializing a build environment).This system has been used by the researcher on many previous projects for building versions of the Android operating system. The source code for the three versions of Android represented in this research was downloaded using the repo tool as specified by the instructions online (Android, n.d., Downloading the source tree). The three different versions of Android were compiled for the three specific devices and installed using the fastboot tool, as

specified online (Android, n.d., Building for devices). This workstation was also used for writing Android update packages.

A laptop was also available. This machine also ran Ubuntu 10.04 and was used to write Android applications. This is a 32-bit machine that has been used by the researcher in the past to develop Android applications using the Android software development kit. As specified online, the Eclipse development environment, the Android Developer Tools, and the Android Software Development Kit were installed on this laptop (Android, n.d., Get the Android SDK).

The Android build for each device included certain private keys which were stored on the workstation. As stated in Chapter 2, The OEM is capable of designing official system updates. The OEM maintains the private key for signing these update packages, so one can reason that the OEM is also capable of designing the update packages used in this study. It is highly unlikely that the OEM would ever release updates created with malicious intent. If the private key were to compromised, either by a network intrusion or a rouge employee, or if the private key were reverse engineered, a trained individual is capable of creating these update packages.

3.3 Specific Update Packages

Four specific update packages each representing a specific task were created for each device, resulting in twelve total packages. Though each specific task was packaged differently for each device, the packaging process was very similar from device to device. This means that the package used to complete one goal on the Nexus One was similar to the packages to complete the same goal delivered to the Nexus S and the Xoom. The first two of these packages represent tasks which would be beneficial for law enforcement.

The second two represent malicious attacks on the user. Criteria for success were defined for each update package. If the update package executed as specified, it was executed successfully. Any other results were considered a failure.

3.3.1 Forensic Image of the Device

The first package delivered to each phone was designed to create a forensic image of the device. The package contained an application written using the Android software development kit, plus a root exploit. The applications were all written for the 2.3 version of Android, as all of the devices ran Android 2.3 or above. The package first rooted the device using the packaged exploit, then placed the application in the /system/app directory. Application files placed here are automatically installed upon boot. This application, requiring root privileges, was designed to start at boot and create a bitstream image of the device's userdata partition and place this image on the device's external storage media.

3.3.2 Limited Extraction of Data

The second package was designed to extract a limited set of data. This was designed to extract only the device's web browsing history. This update installed an agent on the device which, at boot, used the Android API to extract history from the device's web browser and write to the SD card. This is similar to how the current Android forensic tools in the study in Hoog's *Android Forensics: Investigation, Analysis, and Mobile Security for Google Android* operate. The forensic tools examined in the study install agent applications on the device which leverage the device’s API to perform an extraction of data (Hoog, 2011). This update application also installs an application which leverages the device’s API, but it is delivered via an update package, which is not

how the tools in this study are delivered. Unlike those tools, the application written for this study was only focused on extracting web history.

3.3.3 Evidence Planting

The third package is the first one which is malicious to the user. The researcher needed a copy of the device's web browsing history file. A bitstream image of the userdata partition was created using the first update package, if the first update package was successfully executed. The researcher had this image available and needed a web browsing history file from the device, so the researcher extracted the web browsing history file from this image. (If the first update package was not completed successfully, this file could be obtained in another fashion. This first update package included rooting the device, and if the device is rooted then individual files or an entire image of the device can be extracted (Lessard & Kessler, 2010)). The researcher extracted the device's web browsing history and inserted browsing history entries of the device browsing three websites which the device did not browse to. This modified web browsing history file was placed into an update package. The update package mounted the userdata partition as read/write, then overwrote the web browsing history with this file. This update package represents malicious evidence planting designed to make it appear as though the user browsed to websites with suspicious, or even illegal, content.

3.3.4 Malware Installation

An application representing a piece of malware was written using the Android software development kit. This piece of malware was designed to start at boot and download a file from the Internet. The application was designed to download the file to the device without the user's expressed consent. The update package placed this

malicious application in the /system/app directory, and when the phone booted, this application was installed. This represented malware on the device which uses the network connectivity without the user's knowledge, and this can cost the user money if the file is downloaded over the device's 3G or 4G connection.

3.4 Detailed Methodology

The following sections discuss the detailed methods used in this research, along with what data was collected.

3.4.1 Building the Operating System

Using the research computer, three builds of the Android operating system were made. They were Gingerbread, or Android 2.3, for the Nexus One, Ice Cream Sandwich, or Android 4.0, for the Nexus S, and Jelly Bean, or Android 4.1, for the Xoom. These were built for "user," as opposed to for "userdebug." The latter is for research purposes and the resulting build includes root privileges; the former represents a build similar to what would be found on a device sold in a store, with no root privileges. The process of downloading the source and building for a device followed the documentation as discussed in Chapter 2. The researcher noted the public keys included in each build in the /system/etc/security/otacerts.zip file as discussed in Chapter 2 and ensured that the matching private keys were available.

3.4.2 Building the Update Packages

The four update packages described in the previous section were created for each of the three devices, resulting in twelve update packages. Each package was signed with a private key using the RSA algorithm as discussed in Chapter 2. The exact signing statement was as follows:

```
java -Xmx1024m -jar signapk.jar -w testkey.x509.pem testkey.pk8 update.zip
update-signed.zip
```

In this script, signapk.jar is a Java executable jar which can sign updates or applications as needed. The testkey.x509.pem and testkey.pk8 represent the private certificate to sign the update, update.zip is the original unsigned package, and update-signed.zip is the resulting signed update.

3.4.3 Executing the Update Packages

For each update package and each device, the device must be prepared and populated with a specific set of data. The phone was reset to factory state and then populated with a specific set of data so the device remained in a known state before each update package execution. The update package was added to the device. The device was powered off and booted into recovery mode, where the update was executed. A specific set of criteria, written in the following sections, was defined for each update package; if these criteria were met, the update was executed successfully, and if any other results were noted, the update was not executed successfully. The device was powered back on and analyzed to see if the update was successfully executed. Each execution was noted as successful or non-successful, and if the update was executed unsuccessfully, notes regarding the failure were taken. The following chart visualizes this process.

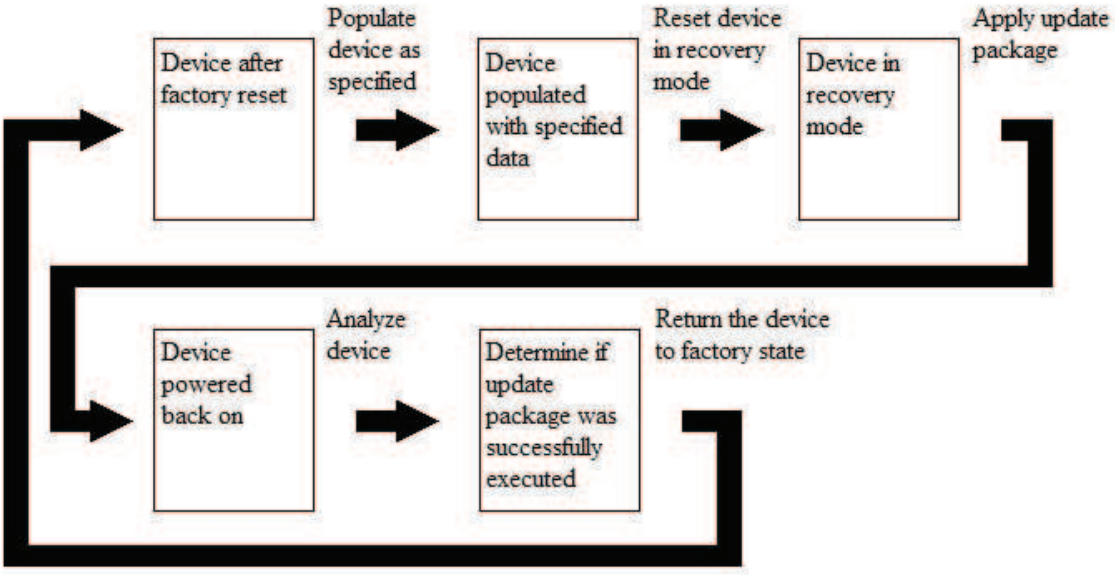

Figure 3.2 Chart of Methodology for One Device and One Package

For each of the four updates and three devices, this cycle was repeated five times for a set of 60 executions. The cycle was repeated multiple times in order to observe a trend of results instead of simply an anomaly. The following sections describe the population of each device and specific success criteria for each update package.

3.4.3.1 Forensic Image of the Device

Before executing the update package to create a forensic image of the userdata partition of the device, the device must be in a known state. The device was reset to factory state. The applications Astro File Browser and the Holy Bible were installed on the device. The web browser browsed to the websites www.purdue.edu, developer.android.com, and www.nfl.com.

The update package was loaded to the external storage media, and then the device was powered off powered into recovery mode where the update package was installed.

The device was powered back on and analyzed for correct execution of the update package. For the execution to be considered a successful execution of the update package, the following criteria must be met:

- The application creating the bitstream image must open upon and begin imaging upon boot
- A bitstream image of the userdata partition must be on the external storage media
- The bistream image must be of the correct size for the device
- The bitstream image must contain a valid file system with no corruptions
- Astro File Browser app and the Holy Bible app must be in the bitstream image
- The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com

The Nexus One's userdata partition is formatted as YAFFS2 and required the unyaffs2 tool to extract the file system (Penguin.lin, 2012). The current version was used, which was released in August 2012 and was downloaded from Google Code. The other two devices' userdata partitions are formatted ext4 and were examined in FTK Imager version 3.1.1.

If all of these conditions were correct upon execution of the update package, then the update was executed successfully. If any other result were found, the update was not executed successfully, and any anomalies were noted.

3.4.3.2 Limited Extraction of Data

Before executing the update package to extract a limited set of data, or the web history, the device must first be in a known state. The device was reset to a factory state,

and the web browser browsed to the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com.

The update package was loaded to the external storage media, and then the device was powered off powered into recovery mode where the update package was installed. The device was powered back on and analyzed for correct execution of the update package.

If the package was executed successfully, the following criteria must be met:

- A CSV file must be on the device's external storage media containing the web browsing history
- The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com

If these criteria were met, the package was executed successfully; if any other results occurred, the update was not executed successfully, and any anomalies were be noted.

3.4.3.3 Evidence Planting

Before executing the update package to extract a limited set of data, or the web history, the device must first be in a known state. The device was reset to a factory state, and the web browser browsed to the websites from the first test. The web history file from the bistream image from the first test for each device was extracted and stored for this update package; if the first test was not executed successfully, this file could be retrieved in a different fashion. This file was modified to include three extra items in the web browsing history to fake the browser navigating to these pages. These three extra webpages were www.bengals.com, www.microsoft.com, and www.walmart.com.

The update package was loaded to the device's external storage, and then the device was powered off and powered into recovery mode where the update package was installed. The device was powered back on and analyzed for correct execution of the update package.

If the update package was executed successfully, the following criteria must be met:

- The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package
- The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com
- The browser must open and execute properly

If these criteria were met, the package was executed successfully; if any of these criteria were not met, the update package was not executed successfully.

3.4.3.4 Malware Installation

Before executing the update package to load some malware onto the device, the device must be in a known state. The device was reset to factory state and the app Astro File Browser was installed on the device. This app is a file browser; it allowed the researcher to see if the file successfully downloaded and its size.

A 32 megabyte file called "spaceWaster.txt" was loaded onto a local web server. The MD5 hash of this file was noted. The application representing malware in this update package downloaded this file via http. The update package was loaded to the device's external storage. The device was powered off, booted into recovery mode, and the update package was installed. The device was powered back on and analyzed for correct

execution of the update package. If the package was executed successfully, the following criteria must be met:

- The file “spaceWaster.txt” must be on the device’s external storage
- “spaceWaster.txt” must be 32 megabytes
- The MD5 hash of “spaceWaster.txt” must match the original

If these criteria were met perfectly then this update package was successfully executed; if any other results were found, this update was not executed successfully.

3.4.4 Analysis

When completed, sixty tests have been executed, each yielding results of “successful” or “not successful” with an explanation. These results consist of five repetitions each of four update packages on three devices. These results were stored in table form. A table was made for each execution, or 60 total such tables. Another table was made for each set of five repetitions of an update package executing on a specific device. This results in 12 such tables. Trends in the data in terms of success on specific devices and success on specific update packages was noted.

Also, a discussion of soundness of the update package is in the final chapter. This consists of the size of the update package, the files added to the device and their sizes, what all files were modified or created as a result of execution of the update package, and any other considerations worth noting. For the first two update packages, forensic soundness is an essential consideration. For the final two update packages, an effective malicious tool should leave minimal evidence on a device, so soundness is also an important consideration

CHAPTER 4. FINDINGS

This study was a proof of concept study. The forensic related update packages were not intended to be analyzed as tools to soon use in the field, and the malicious update packages were not intended to represent projected future malware. This was a study in the effectiveness of update packages as a method of deploying utilities to a device. The research methodology detailed in Chapter 3 was executed, and the results of the tests discussed in the methodology are organized in 12 tables, which summarize the 60 total executions as described in Chapter 3. The complete data for each of the 60 executions is in the appendix.

In all sets of five repetitions of one update package on one device, either a set of five failures or a set of five successes was noted. The Nexus One succeeded in all executions of all four update packages. The Nexus S and the Xoom both successfully executed Update package 2, or the limited extraction update package, but every other update package was noted as an unsuccessful execution. The overall success rate was 50%.

4.1 Forensic Image of the Device

The first update package, a forensic image of the device, was attempted five times each on the three different devices, an HTC Nexus One running the Gingerbread build of

Android, a Samsung Nexus S running the Ice Cream Sandwich build of Android, and a Motorola Xoom running the Jelly Bean build of Android. The following three tables contain the results of the five tests on these three devices and are each followed by detailed explanations of the data. Only the Nexus One successfully executed this update package. The observations of both successfully met and failed requirements are in the details in the following sections.

Table 4.1 *Results for Update Package 1 on Nexus One*

	Results (Success or Fail)	Notes
Execution 1	Success	Image created successfully, both apps present, web history present
Execution 2	Success	Same as 1 above
Execution 3	Success	Same as 1 above
Execution 4	Success	Same as 1 above
Execution 5	Success	Same as 1 above

For each execution of this update package, the device was set to a factory state, populated with a specific set of data, and rebooted into recovery mode where the researcher applied the update package. In all five cases the update package was installed with no errors. The Nexus One was then rebooted and the observations were compared to the criteria for success established in Chapter 3.

Upon reboot, the imaging app opened up and began creating the image and storing it to the external storage card. The first requirement was satisfied. The image

was finished after a few minutes and was stored entirely on the storage card, which satisfied the second requirement. The image was of size 196.2 Megabytes. The device reported its size as 196 Megabytes in the application management menu in settings, and this number was rounded to the nearest Megabyte. This satisfies the third requirement.

The researcher attempted to use the unyaffs tool and many other freely available tools to browse the image of the device. It was formatted yaffs2, which does not currently have support in FTK Imager. According to Quick and Alzaabi (2011), a bitstream image of a yaffs2 partition, such as on Android,"cannot be realistically used to retrieve files." This presented a difficulty in satisfying this requirement. The researcher was unable to extract files from the image but was able to search through the image using methods presented in Lessard and Kessler (2010). The researcher could not absolutely confirm that the image had no corruptions, but there has been no evidence to suggest contrary. The researcher realized after reviewing Quick and Alzaabi (2011), which was after collecting data, that a better tool would have both collected the bitstream image that was collected and also a NANDump, as described in that paper. This tool would have provided an easier method of exploring the extracted files, though it would not be a bitstream image and thus would not include all slack space. Obtaining both would be the proper step if these tools are to be edited in the future.

The researcher used methods presented in Lessard and Kessler (2010) to manually search through the image. The researcher extracted fragments of the application files and pieced the files together. This is an extremely time consuming process. The application files, Astro File Browser and the Holy Bible, are 2.3 Megabytes and 6.7 Megabytes respectively. These files were fragmented in the image into sections ranging from four

kilobytes to 76 kilobytes. This made piecing the files together extremely time consuming. The researcher successfully pieced together the Astro File Browser app in the first image. The researcher pieced together the beginning and the end of the app install files in all five images. This does not absolutely confirm the files existed in the image, but the researcher has been given no indication to believe this not to be the case. The researcher considers this requirement satisfied on all five repetitions.

The same manual process was used to find web browsing history. Evidence of browsing to www.purdue.edu, www.nfl.com, and developer.android.com were found in all five images. The following image is an example of this evidence, pulled from a Hex editor, similar to the process in Lessard and Kessler (2010).

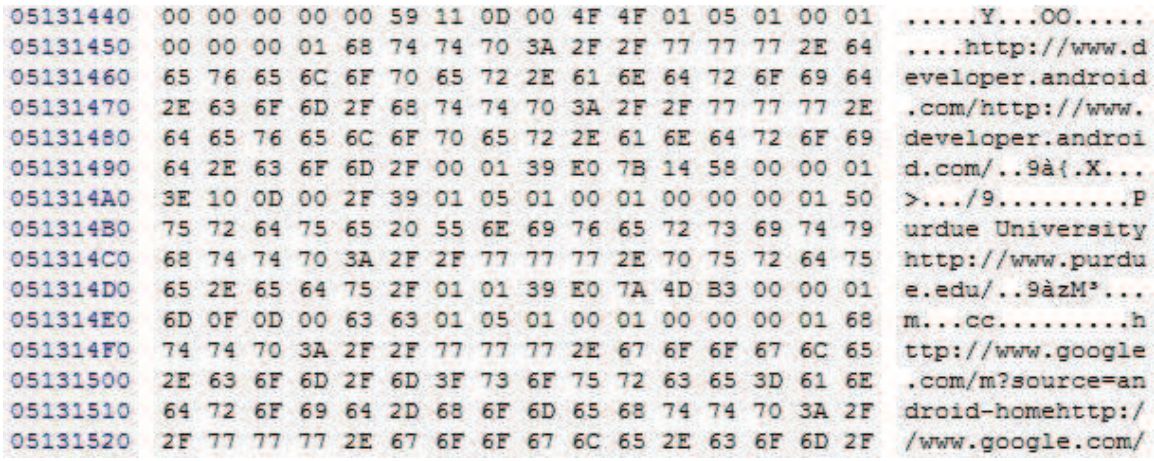

Figure 4.1 Web History in Bitstream Image of Nexus One

The above image shows two of the URLs within the image file viewable in a Hex editor, and similar evidence can be found for the third URL, www.nfl.com. Such

evidence was found in all five images. The researcher considers this requirement to also be satisfied.

All six requirements were satisfied in all five executions. The researcher was unable to find a tool to work properly with the image that was created in order to extract the file system. If such a tool were available this would have been far simpler and less time consuming to complete. Such a tool was not available, so the method of parsing through the bitstream image with a hex editor presented in Kessler and Lessard (2010) was used. A future version of the tool for other YAFFS2 devices should incorporate the NANDump tool as presented in Quick and Alzaabi (2011).

Table 4.2 *Results for Update Package 1 on Nexus S*

	Results (Success or Fail)	Notes
Execution 1	Fail	Imaging application did not start automatically. Upon manually opening the application and then restarting the device the application opened automatically and successfully created the image.
Execution 2	Fail	Same as 1 above
Execution 3	Fail	Same as 1 above
Execution 4	Fail	Same as 1 above
Execution 5	Fail	Same as 1 above

The device was rebooted after the update package installed with no errors. The imaging app did not start automatically. The researcher manually opened up the imaging app. Upon subsequent reboots, the imaging app opened automatically. It required being manually launched first to open automatically. This first requirement was recorded as a failure.

The imaging app created an image of the userdata partition on the device's external storage partition. This satisfies the second requirement. The image created was one Gigabyte, which is a match to the device's actual userdata partition size. This satisfies the third requirement.

The image was successfully loaded into FTK Imager, which identified the image as being formatted ext4. This satisfies the fourth requirement. The file system contained the Astro File Browser app and the Holy Bible app, which satisfies the fifth requirement. The following screenshot shows these two files in an image from the Nexus S. The directory open in the screenshot is /data/app, where installed application install files are stored.

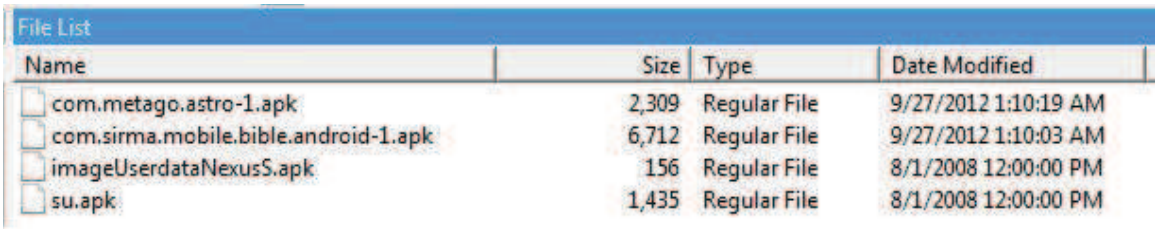

Figure 4.2 Apps Directory in Bitstream Image of Nexus S

The file su.apk is a component of rooting the device, which was part of the update package. The file imageUserdataNexusS.apk was the application file designed to image

the device, and the other two files were part of populating the device. An interesting observation is the time stamp. The modified data of the two apps that were added as part of populating the device reflect when the apps were installed on the device. The other two apps were part of the update package and were installed via recovery mode. The time stamp for both as viewed in FTK Imager is 8/1/2008 at noon.

The web browsing history was intact in the image and contained the three required websites, which satisfies the final requirement. The file where web browsing data was stored was /data/data/com.android.browser/databases/browser2.db and can be viewed in a SQLite3 viewer.

The only failure noted was in the app launching automatically upon boot. It failed to do so, and thus the execution was labeled a failure. These observations occurred in all five executions of the first update package on the Nexus S.

Table 4.3 *Results for Update Package 1 on Xoom*

	Results (Success or Fail)	Notes
Execution 1	Fail	Device could not detect SD card; Could not create an image
Execution 2	Fail	Same as 1 above
Execution 3	Fail	Same as 1 above
Execution 4	Fail	Same as 1 above
Execution 5	Fail	Same as 1 above

The imaging app did not automatically launch upon boot, similarly to the behaviors of the Nexus S version of this update package. If the app were to be manually opened, it would automatically launch on subsequent boots, also similarly to the behaviors of the Nexus S version of this package. The first requirement was not successfully met.

The device could not identify the external storage media, which was an SD card. The researcher mounted a USB flash drive to the device, but this also was not recognized. The build of Android may have been corrupted, or perhaps the hardware itself was damaged. In either case, requirement number two could not be satisfied. All subsequent requirements depend upon this requirement being satisfied, so no requirements were met and the execution was recorded as a failure. These behaviors were observed in all five executions of this update package on the Xoom.

The researcher did create a bitstream image of the device via ADB passed the image directly from the device to the computer; this was performed manually. The web history file necessary for Update Package 3 was extracted from this manually created image. This image contained the proper apps and web history.

The cause for the device not recognizing external storage is not entirely known. Either the device could be physically damaged or the build the researcher created for this study could be corrupted. In either case, the execution was unsuccessful, both for this failure and for the app not launching automatically upon boot.

4.2 Limited Extraction of Data

The second update package, an extraction of web history using the developer API, was attempted five times each on the three devices, just as with the previous update

package. The following three tables contain the results of the five tests on these three devices and are each followed by detailed explanations of the data. All three devices successfully executed this update package. Details regarding these successful executions are in the following sections.

Table 4.4 *Results for Update Package 2 on Nexus One*

	Results (Success or Fail)	Notes
Execution 1	Success	Displayed multiple entries for each of the four websites; perhaps because two were redirected to mobile version of the site, and mistyped some if not all of them; pages designated as "favorites" by Android also were in here
Execution 2	Success	Displayed multiple entries for only two, these two redirected to mobile version of the site. No mistypes. Again had favorites.
Execution 3	Success	Same as 2 above
Execution 4	Success	Some extraneous entries due to mistypes
Execution 5	Success	Same as 2 and 3 above

In all five executions, a CSV file was created and stored on the device's storage card following execution. This satisfies the first requirement. In all five cases, this CSV file contained the web history of the device which included all of the required websites. The website www.att.com was redirected to www.m.att.com, a mobile version of the website, and both of these appeared in all five files. The websites www.t-mobile.com and www.m.t-mobile.com both appeared for the same reason. The history also included pages pre-selected as favorites, including CNN, Yahoo, ESPN, and others. In some cases, extra entries were included when the researcher mistyped a page on the device's touchscreen. The errors in typing and the redirection to the mobile versions of websites are not reasons to consider the second requirement a failure. The Nexus One succeeded in all five repetitions.

Table 4.5 *Results for Update Package 2 on Nexus S*

	Results (Success or Fail)	Notes
Execution 1	Success	Displayed multiple entries for some of the websites, perhaps because of redirection to mobile site
Execution 2	Success	Same as 1 above
Execution 3	Success	Same as 1 above
Execution 4	Success	Same as 1 above
Execution 5	Success	Same as 1 above

All five executions of this update package succeeded as both requirements were met. The file was stored on the device's /sdcard partition, which is an internal partition of storage that acts like removable media. There were redirections to the mobile version of websites similarly to the Nexus One previously, but there were no mistypes on the Nexus S.

Table 4.6 *Results for Update Package 2 on Xoom*

	Results (Success or Fail)	Notes
Execution 1	Success	Exactly as specified, no repeat websites
Execution 2	Success	Same as 1 above
Execution 3	Success	Same as 1 above
Execution 4	Success	Same as 1 above
Execution 5	Success	Same as 1 above

In all five executions, a CSV file was created and stored in a directory used for external storage. This satisfies the first requirement. The researcher did not type any websites incorrectly. The Xoom did not redirect to the mobile versions of any websites so this CVS file did not contain any duplicate entries. All five executions were considered successes.

4.3 Evidence Planting

The third update package, a malicious package which plants websites in history which the user did not visit, was attempted five times each on the three devices. The following three tables contain the results of the five tests on these three devices and are

followed with further information about the data. The Nexus One was the only device to successfully execute this update package. The Nexus S and the Xoom both failed, though different observations were noted for each during these executions.

Table 4.7 *Results for Update Package 3 on Nexus One*

	Results (Success or Fail)	Notes
Execution 1	Success	All three inserted web pages appear in browsing history
Execution 2	Success	Same as 1 above
Execution 3	Success	Same as 1 above
Execution 4	Success	Same as 1 above
Execution 5	Success	Same as 1 above

The first requirement includes websites which were populated before the update was applied. These three appeared in the device's web browsing history upon rebooting, which means the first requirement was successfully met on the Nexus One. The second requirement contains websites which were to be added as a result of applying the update package. These three websites also appeared in web browsing history, even though the device did not actually navigate to these websites. The second requirement was successfully met. The web browser opened properly without crashes and reported web history properly, which satisfies the third requirement. These behaviors were observed in all five executions, so all five executions on the Nexus One were successes.

As stated in update package 1, extracting individual files from the Nexus One image was unrealistic. The web history file was obtained from the device after update

package 1 was executed by using the Android Debug Bridge. The device was rooted, so with root access via the Android Debug Bridge the file was obtained and modified for this update package.

Table 4.8 *Results for Update Package 3 on Nexus S*

	Results (Success or Fail)	Notes
Execution 1	Fail	Browser crashed and history was cleared
Execution 2	Fail	Same as 1 above
Execution 3	Fail	Same as 1 above
Execution 4	Fail	Same as 1 above
Execution 5	Fail	Same as 1 above

Upon reboot, the researcher opened the web browser to see if all six required web pages were in web history. In all five executions, the browser app crashed upon opening. It was opened again and all history was gone. This behavior means all three requirements were failed in all five repetitions on the Nexus S.

Table 4.9 *Results for Update Package 3 on Xoom*

	Results (Success or Fail)	Notes
Execution 1	Fail	No change, history remained intact, browser did not crash
Execution 2	Fail	Same as 1 above
Execution 3	Fail	Same as 1 above
Execution 4	Fail	Same as 1 above
Execution 5	Fail	Same as 1 above

Upon reboot after applying the update package, the researcher opened the web browser to see if all six required web pages were in history. The web browsing app opened properly. The original three webpages of the first requirement were in history, so this requirement was satisfied. The webpages in the second requirement, which were planted in the update package, were not in web browsing history, so the second requirement was not successfully met. The browser opened properly with no crashes, so the third requirement was met. In all five executions, the Xoom did not successfully execute the update package since the planted web pages did not appear in history.

4.4 Malware Installation

The final update package, a malicious package which installs a malicious app that downloads a file to the device without the user's permission, was attempted five times each on the three devices. The following three tables contain the results of the five tests on these three devices and are each followed with further information regarding the results. The Nexus One successfully executed all five repetitions. The Nexus S and the

Xoom both failed all five of their respective repetitions. Details regarding the successes of the Nexus One repetitions and the failures of the other two devices are in the following sections.

Table 4.10 *Results for Update Package 4 on Nexus One*

	Results (Success or Fail)	Notes
Execution 1	Success	Hash of downloaded file: 279e4d151bcf4cbd9e2aab738b230aa7
Execution 2	Success	Same hash as 1 above
Execution 3	Success	Same hash as 1 above
Execution 4	Success	Same hash as 1 above
Execution 5	Success	Same hash as 1 above

The file "spaceWaster.txt" was loaded onto a local webserver. The MD5 hash of this file was 279e4d151bcf4cbd9e2aab738b230aa7. This file was downloaded to the device's storage card and was of size 32 megabytes. The first two requirements were successfully met by these behaviors. The MD5 hash of the downloaded file was an exact match to that on the web server, which satisfies the third requirement. These behaviors were observed in all five executions, so the Nexus One succeeded in all five repetitions.

Table 4.11 *Results for Update Package 4 on Nexus S*

	Results (Success or Fail)	Notes
Execution 1	Fail	App was successfully installed but immediately crashed upon opening
Execution 2	Fail	Same as 1 above
Execution 3	Fail	Same as 1 above
Execution 4	Fail	Same as 1 above
Execution 5	Fail	Same as 1 above

The app was on the device upon reboot. It was launched but it immediately crashed. It did not download any file. This behaviors means the Nexus S failed on all three requirements and this happened in all five executions.

Table 4.12 *Results for Update Package 4 on Xoom*

	Results (Success or Fail)	Notes
Execution 1	Fail	App was successfully installed but immediately crashed upon opening
Execution 2	Fail	Same as 1 above
Execution 3	Fail	Same as 1 above
Execution 4	Fail	Same as 1 above
Execution 5	Fail	Same as 1 above

The behaviors noted in the Nexus S were repeated on the Xoom. None of the three requirements for success were met in any of the five repetitions. All five executions were noted as unsuccessful.

4.5 Summary of Results

Of the 60 tests performed, 30 tests resulted successfully, or an even 50% success rate. Five repetitions were performed each on 12 tests, and each set of five repetitions yielded identical results, meaning these results demonstrated remarkable consistency. Either the test was performed successfully or it failed, and it worked that way each and every time for the same tests on the same device running the same version of the Android operating system. The following tables contain the data organized by update package.

Table 4.13 *Results for Update Package 1 across all devices*

	Successes (out of five repetitions)	Success percentage
Nexus One (Gingerbread)	5	100
Nexus S (Ice Cream Sandwich)	0	0
Xoom (Jelly Bean)	0	0

Update package 1, or imaging the device, worked as designed on the Nexus One for all repetitions. The Nexus S failed as the app did not launch at boot. The implication for this is important. If a device is password locked and an investigator wishes to obtain a bitstream image of the device, one potential method would be to install an app to image

the device via recovery mode. If the device is password locked, the investigator would be unable to manually open the app. The app did not launch automatically and in the case of the locked phone, the investigator would have no way of getting the app to launch and therefore no way to create the bitstream image using this update package.

The Xoom failed all five repetitions for both the same reason as the Nexus S and also to a technical issue. The device was unable to locate an external storage card and thus had no ability to save an image to an external storage card.

Table 4.14 *Results for Update Package 2 across all devices*

	Successes (out of five repetitions)	Success percentage
Nexus One (Gingerbread)	5	100
Nexus S (Ice Cream Sandwich)	5	100
Xoom (Jelly Bean)	5	100

Update package 2, or extraction of web history, worked successfully on all repetitions of all three devices. This was the only update package to have this level of success.

Table 4.15 *Results for Update Package 3 across all devices*

	Successes (out of five repetitions)	Success percentage
Nexus One (Gingerbread)	5	100
Nexus S (Ice Cream Sandwich)	0	0
Xoom (Jelly Bean)	0	0

Update package 3, or evidence planting, worked successfully on the Nexus One. The other two devices both failed but in different ways. The Nexus S web browser crashed upon reboot after installing the update and all web history was purged. The Xoom web browser retained the original populated web browsing history and did not crash upon boot but did not have the planted web browsing history.

Table 4.16 *Results for Update Package 4 across all devices*

	Successes (out of five repetitions)	Success percentage
Nexus One (Gingerbread)	5	100
Nexus S (Ice Cream Sandwich)	0	0
Xoom (Jelly Bean)	0	0

The Nexus One successfully executed Update package 4, or malware installation. The Nexus S and the Xoom both failed in the same way. The malware app crashed as soon as it was opened on both devices.

Update package 2 was the only one that succeeded on all three devices. Update package 4 failed on the Nexus S and the Xoom in identical fashions, Update package 3 failed on the Nexus S and the Xoom in different fashions, and Update package 1 failed on the Xoom for the same reason as it failed on the Nexus S but also had an additional technical issue.

The following three tables display the same data but organized by device.

Table 4.17 *Results for Update Packages on Nexus One*

	Successes (out of five repetitions)	Success percentage
Update Package 1	5	100
Update Package 2	5	100
Update Package 3	5	100
Update Package 4	5	100

The Nexus One succeeded in all update packages. This device ran Gingerbread, or the oldest version of Android represented. All update packages were successfully installed in recovery mode. Update packages 1, 2, and 4 involved installing an app and in all three cases the app installed properly on the device.

Table 4.18 *Results for Update Packages on Nexus S*

	Successes (out of five repetitions)	Success percentage
Update Package 1	0	0
Update Package 2	5	100
Update Package 3	0	0
Update Package 4	0	0

The Nexus S failed all update packages expect for the second one, representing a limited extraction of data. This update package leveraged the device's API to output web history. This process is documented on the Android development website (Android, 2013, Browser). The other update packages were different in that they relied on methods not preferred on the Android development website, including "hacks," for success. Update package 1 relied upon rooting the device, which is considered by the hacking community to be a hack (Lessard & Kessler, 2010), and executing a shell command using the "dd" utility.

Update package 3 relied upon overwriting in recovery mode a database which the browser app uses. This is opposite of Update package 2, which relied upon the device's API to access web history and, as previously stated, is documented on the Android development website.

Update package 4 relied upon downloading a file in the background without the user's permission or knowledge. This did not leverage the device's browser. The standard method of downloading files in Android is to use the DownloadManager utility

(Android, 2013, DownloadManager), but this notifies the user of a download. A non-preferred method of downloading was used, so this update package, along with the first and third, used methods which are not documented as preferred methods on the Android development website.

Table 4.19 *Results for Update Packages on Xoom*

	Successes (out of five repetitions)	Success percentage
Update Package 1	0	0
Update Package 2	5	100
Update Package 3	0	0
Update Package 4	0	0

The Xoom, like the Nexus S, failed on all four update packages with the exception of Update Package 2. The previous section discussed how update packages 1, 3, and 4 used non-preferred methods to achieve success, including some hacks. The same is true to these update packages. The Xoom's only successful update package was Update package 2, like the Nexus S, which is the only update package to rely on documented methods on the Android development website to successfully execute.

4.6 Size of the Apps

All of the update packages, expect for the one which plants data, involve installing an app. The size of this app needs to be known to understand how much of an impact is made on the device by applying these packages. The following table contains the sizes of all of these apps.

Table 4.20 *Sizes of Apps in Update Packages*

Device	Update Package	App Size (kB)
Nexus One	1 (Image Device)	155.7
Nexus S	1 (Image Device)	155.9
Xoom	1 (Image Device)	155.9
Nexus One	2 (Limited Extraction)	155.7
Nexus S	2 (Limited Extraction)	155.7
Xoom	2 (Limited Extraction)	155.7
Nexus One	4 (Malware Installation)	156.5
Nexus S	4 (Malware Installation)	156.5
Xoom	4 (Malware Installation)	156.5

All of these apps are small and have a small footprint when compared to other apps on the Play Store. ABI Research studied the size of both iOS and Android apps in 2012. The study stated the average app size for Android was around six megabytes as of September 2012, and this represented a 10% increase since March 2012 (ABI Research, 2012). The apps designed for this thesis are all very simple, use no graphics, and were designed to have as small a footprint as possible. Also of note is that the first update packages required rooting the device. The su binary, placed in the system partition, is 371 KB, and the su app, which was downloaded from the Android Market, is 1.4 MB. These files needed to be placed on the device for this update to work, and they can both also be uninstalled easily with another update package. A previous section suggested that the Nexus One version of update package 1 should be revised to include the NANDump

tool, so this would also have be to packaged in future releases and considered in a soundness discussion.

The first update package changed the device since it installed the app designed for to create a bitstream image of the device, the su app, and the su binary. The second update package installed the app designed to extract web history, but no other files were changed. The third update package modified the web browsing file by adding some extra entries. The fourth update package installed the app designed to represent malware and attempted to download the spaceWaster.txt file. Beyond these described changes and any system logs which were created as a result of running this update packages, no other files were created or modified by these update packages.

CHAPTER 5. DISCUSSION AND CONCLUSIONS

Many of the update packages failed, and when an update package for a device failed once, it failed in all five executions in the exact same way. The Nexus One was the only device to have complete success. The other two failed in all but one test. The key difference between the successes of the Nexus One and the two other devices, the Nexus S and the Xoom, is the operating system. The Nexus One runs the oldest version of the Android operating system.

This proof of concept study aimed to determine if update packages could be used in a forensic or a malicious method on Android devices. The results of this study do not indicate that these tools are ready to be used. More research into the differences between versions of the Android operating system and differences between data storage of different versions of common apps is required to move this research forward. In all 60 executions in this research, the update was installed in recovery mode with no errors, but the criteria for success as defined in Chapter 3 were only met 50% of the time.

This chapter focuses on analyzing the data presented in the previous chapter. Trends are discussed, along with an analysis of the differing results from one device to the next. Potential future research is presented.

5.1 Answering the Research Question

Chapter 1 proposed the following research question: can update packages be used to deliver malicious payloads and/or utilities beneficial for a forensic investigation on current Android devices? The answer is no after viewing the results and their context. Three of the four update packages tested relied on some hacks and these have been stopped by the security mechanisms of the newer operating systems. With some further research into these security mechanisms and different versions of key common apps, such as the web browser, these update packages could be revised and retested for success.

5.2 Explanations of Different Results for the Devices

As discussed in the previous chapter, there are some trends in this data. The Nexus One successfully executed all update packages. The Nexus S and the Xoom, which ran newer versions of the operating system than the build of Android on the Nexus One, succeeded only on update package 2. The other three update packages relied on hacks, as discussed in Chapter 4.

In all executions, the update package was installed in recovery mode with no errors. In no case did the recovery mode reject an update package. In all cases are an app was installed on the device as a result of the update package, the app appeared in the device's app list upon reboot.

The consistency within each set of update packages on a device was notable. In all 12 sets of a specific update package on a specific device, all five repetitions either succeeded or failed in the same way. The devices were all populated to a known set of data. In each case where websites were visited, the same set of websites was visited each time. When apps were installed, the same set of apps were installed each time. The

conditions for each set of five repetitions of one update package on one device were as identical as possible. This remove any variables which might have led to differing results within each set of an update package and a device. The results were same for each set of five repetitions, so these update packages execute consistently. If these packages are to be improved and eventually used as forensic tools, this consistency satisfies a Daubert requirement. One of the Daubert criteria for admissibility of scientific evidence is the reliability of the forensic process (Project on Scientific Knowledge and Public Policy, 2003).

The Nexus One successfully executed all four update packages, but the Nexus S and the Xoom failed the same three update packages. There are two potential variables that exist to cause this difference. The Nexus One could have hardware that is better suited for these update packages, or the newer versions of the operating system could contain features to prevent these update packages from executing properly.

The first possibility is the Nexus One device itself is the cause for better execution of update packages. The Nexus S and the Xoom both contain some hardware upgrades from the Nexus One, and all three are created by different manufacturers. The other possibility of the difference of results between the Nexus One and the other two devices is the operating system difference. The Nexus One ran Gingerbread, or Android 2.3, while the newer two ran Ice Cream Sandwich and Jellybean, or 4.0 and 4.1.

The second possibility is more likely. The update packages, with the exception of the evidence planting package, installed an app on the device. These apps are designed not for a specific platform but for an Android version.

Another consideration is a study of known exploits for Android. Some exploits are targeted at an application, such as a known Adobe Flash Player for Android exploit. According to Adobe, this exploit "could cause a crash and potentially allow an attacker to take control of the affected system" (Adobe, 2013). Some exploits are targeted at a version of Android, and as newer versions of Android are released these vulnerabilities are addressed (Google, 2011). And finally, there are some exploits that are targeted at a lower level of the device and can be an exploit against a specific device instead of a version of the operating system; such an example is a kernel exploit against the Samsung Galaxy S3 (Dawson, 2012).

Device specific exploits tend to be lower level exploits, such as the aforementioned Galaxy S3 exploit. The update packages used in this study were written as apps, except for the evidence planting package, and were delivered to the device via update packages. These did not attempt to write to device files or gain privilege escalation like the Galaxy S3 exploit (Dawson, 2012), except the first update package could only work if it rooted the device. The device imaging app did work at a low level, but that only read from a device block instead of attempting to write to one; that read process required root access so the update package rooted the device for this purpose. The fact that these apps work at a level above the kernel suggests that the explanation of device differences, instead of operating system differences, is an unlikely explanation for the results.

Some exploits are targeted specifically at a certain version of the operating system as stated previously. The upgrade from Gingerbread to Ice Cream Sandwich addressed some security issues, including some exploits for rooting devices. Gingerbreak is an exploit to root the device which works on Gingerbread devices but does not work in Ice

Cream Sandwich devices (Xda Developers, 2011, April 21). ZergRush is a similar exploit that works on Froyo, or Android 2.2, devices and Gingerbread devices but nothing newer (Xda Developers, 2011, October 10). The Nexus One, which was the only device to successfully execute all update packages, ran Gingerbread. The other two devices ran Ice Cream Sandwich and its successor, Jelly Bean. The fact that security was improved at the operating system level from Gingerbread to newer versions suggests that the difference in results lies in the difference in operating systems. Security mechanisms in the newer versions of the operating systems may have prevented some of the "hacks" in the update packages from working.

It is also worthwhile to examine exactly what happened during these failures. In the device imaging update package, the two devices that failed their executions were noted as failures because the app did not open automatically until the user explicitly opened the app. The app installed successfully and was on the home screen but did not open. This is certainly not a hardware related issue. This is likely an operating system issue. The Android operating system provides a framework for applications to run, along with libraries, a runtime environment, and process management (Android, n.d., App Framework). A likely explanation is Ice Cream Sandwich and newer versions of the operating system updated the application framework to forbid an app from detecting the device booting and launching automatically until it has first been launched manually.

A similar explanation can be used for the malware package. This app crashed immediately upon opening. This used an Android class called AsyncTask, which is a class involving thread handling (Android, 2013, AsyncTask). Thread handling also is a

responsibility of the operating system since it provides process management (Android, n.d., App Framework).

A likely explanation for the success of the Nexus One but the failures of the other two is that an improvement in the Android operating system in Ice Cream Sandwich and newer limited some of the tasks that AsyncTask can do. A possibility is that when an AsyncTask attempted to download a file instead of using the included DownloadManager, the operating system shut down the app.

The evidence planting update package is different from the other two. This package updated a database file which contained web browsing history. When the device was rebooted, the web browser application, instead of the rest of the operating system, reacted. The reactions were different in the two failed versions. The web browser on the Nexus S crashed and deleted all web browsing history; the web browser on the Xoom opened but had no visible changes in web browsing history. The web browser, like Android, has multiple versions. The web browser in the Nexus One was version 2.3.1. The browser on the Nexus S was version 4.0.4, and the version on the Xoom was 4.1.1. The cause for the difference in reactions of this update package was the web browser version on the device. The different versions may store data differently and the update packages must be revised from one web browser version to the next.

The cause for the differing results from one device to the next is most likely not a hardware issue. The most likely cause of differing results is the difference in versions of the operating system, or in the evidence planting package, difference in versions of the web browser app. Newer versions of the Android operating system present new security features which limit the functionality of the update packages. Update packages 1, 3, and

4 relied upon hacks. These hacks were allowed to execute on the Nexus One running Gingerbread, but the newer operating systems and the newer web browsers on the other two devices prevented some of this functionality from working. It is possible that these apps can be revised to work on the newer versions of the operating system. It is also possible that some functionality, including the ability to launch an app automatically without first manually opening the app, is not possible in the newer version of the operating system and the requirements for these packages must be revised.

5.3 Results in Context

Some explanations are required for these results to keep the results in proper context. This includes some limitations which must be considered when viewing the results. First, the researcher signed these update packages and deployed them to devices running versions of the operating system which the researcher built. As stated in previous chapters, the researcher acted as an OEM. For these packages to work, the vendors must cooperate. The vendors maintain keys used to sign official packages, and they are certainly capable of cooperating with vendors of forensic tools to sign packages used for a forensic purpose. If the signing keys are not provided, the proper keys must be derived to create update packages for Android devices. Deriving the keys would take reverse engineering a known update package for a specific device using cryptology with the intent of extracting a private key from the files CERT.RSA, CERT.SF, and MANIFEST.MF, as discussed in Chapter 2. If the update package is not properly signed, the default Android recovery mode will reject it.

Second, all of the failures in this thesis can be explained, further research can be applied to improve these results. It must also be noted that the errors experienced were

not related to how the payloads themselves were pushed to the device. If these tools are improved such that the device imaging app opens automatically upon boot and that capability is applied to other forensic update packages, these packages would be an improvement over current forensic tools for Android devices as these tools cannot bypass the device password. Given the minimal impact of these updates, the consistency viewed in the results of executing these updates, and with improvement in the execution of these packages, the update packages, after improvements and validations, can be used in criminal investigations as long as the investigator documents all actions taken. With this consideration applied to the malicious update packages, users with malicious intent can have a harmful tool with a small footprint.

These results were viewed on devices which run Gingerbread, Ice Cream Sandwich, and Jelly Bean. As of press time, the release date and name of the next version of Android is unknown, though the last two and a half years before press date have brought the three versions of Android tested and also Honeycomb (Ducrohet, 2010; Ducrohet, 2011; Ghosh, 2012). These update packages failed on newer version of the operating system while succeeding on the oldest one. One can assume that devices running Gingerbread will become less common in future years as more devices with the newest versions of Android sell. These update packages will need to be updated as new versions of the operating system release.

5.4 Future Research

Some future research has already been mentioned in discussion of failed update packages. The NANDump tool should be researched for effectiveness as a supplement to update package 1 on the Nexus One. The security mechanisms of Android which are

likely to have limited the functionality of these updates should be studied and this knowledge could lead to more effective forensic or malicious update packages. More research into other similar topics can be performed. This method is intended for Android devices. Potential research could be performed if equivalent methods could be used on Windows Mobile devices. The signed update packages rely upon an official signature. Research could be performed on reverse engineering these updates to obtain the original key used to sign the packages.

5.5 Conclusions

This proof of concept study presented a new technique of loading apps or making other changes to Android devices. The actual method of putting the payload onto the device executed every time with no reported errors, though the payloads that were installed only succeeded 50% of the time.

The Nexus One successfully executed all four update packages in all fix executions. Two of these update packages were of a forensic nature and the other two were of a malicious nature. These update packages were signed and executed in recovery mode, and the results of these packages were observed as the device was normally booted. These update packages were ported to the Nexus S and the Xoom, and in both cases the only update package that was successfully executed was a limited extraction of web data. The newer versions of the Android operating system contained functionality that limited a forensic imaging update package and a malware package. The newer version of the web browser similarly limited the functionality of an evidence planting update package. An alternative explanation was the Nexus One device itself was more prone to successful executions than the other two devices. This alternative does not adequately explain the

differences in results when comparing these results in context of known exploits on Android devices.

The researcher concluded that these update packages are not ready to be used as they are in their current state, and further development is required on these forensic and malicious update packages. There is great potential for these update packages as eventual forensic tools, and with that comes potential for this concept to be used as a method of delivering malicious apps. More research into security models and app data storage in the newer versions of the operating system is required to progress these packages beyond their current proof of concept state.

LIST OF REFERENCES

LIST OF REFERENCES

ABI Research. (2012, October 6). Average size of mobile games of iOS increased by a whopping 42% between March and September. Retrieved April 11, 2013 from http://www.abiresearch.com/press/average-size-of-mobile-games-for-ios-increased-by-

Adobe. (2013, January 8). Security updates available for Adobe Flash Player. Retrieved March 31, 2013 from http://www.adobe.com/support/security/bulletins/apsb13-01.html

Android. (2012, July 12). RecoverySystem. Retrieved July 15, 2012 from http://developer.android.com/reference/android/os/RecoverySystem.html

Android. (2013, March 21). AsynchTask. Retrieved March 31, 2013 from http://developer.android.com/reference/android/os/AsyncTask.html

Android. (2013, March 21). Browser. Retrieved March 31, 2013 from http://developer.android.com/reference/android/provider/Browser.html

Android. (2013, March 21). DownloadManager. Retrieved March 31, 2013 from http://developer.android.com/reference/android/app/DownloadManager.html

Android. (n.d.). App framework. Retrieved March 31, 2013 from http://developer.android.com/about/versions/index.html

Android. (n.d.). Building for devices. Retrieved January 16, 2013 from http://source.android.com/source/building-devices.html

Android. (n.d.). Downloading the source tree. Retrieved February 1, 2013 from http://source.android.com/source/downloading.html

Android. (n.d.). Get the Android SDK. Retrieved February 1, 2013 from http://developer.android.com/sdk/index.html

Android. (n.d.). Initializing a build environment. Retrieved February 1, 2013 from http://source.android.com/source/initializing.html

Android. (n.d.). Signing your applications. Retrieved July 17, 2012 from http://developer.android.com/tools/publishing/app-signing.html

Atomicdryad. (2010, October). Signapk. Google Code project. Retrived January 16, 2013 from http://code.google.com/p/signapk/

Boneh, D. (1999). Twenty years of attacks on the RSA cryptosystem. Notices of the American Mathematical Society, 46(2), 203 – 213.

Burnette, E. (2010, August 3). Android sales surge, surpass iPhone (updated). Retrieved 19 February, 2011 from http://www.zdnet.com/blog/burnette/android-sales-surge-surpass-iphone-updated/2019

Cellebrite Mobile Synchronization LTD. (2011). UFED Logical supported phones. Retrieved 20 February, 2011 from http://www.cellebrite.com/forensic-products/ufed-standard-kit/ufed-logical-supported-phones.html

ClockworkMod. (n.d.). ROM Manager. Retrieved January 16, 2013 from http://www.clockworkmod.com/rommanager

Commtouch. (2013, February). Internet Threats Trend Report, February 2013. Retrieved March 4, 2013 from http://www.commtouch.com/uploads/pdf/Commtouch-Internet-Threats-Trend-Report-2013-February.pdf

Daniel, L. (Producer). (2009, September 27). Andrew Hoog of viaForensics talks about android forensics [Audio Podcast]. http://www.blogtalkradio.com/TalkForensics.

Dawson, T. (2012, December 17). PSA: Samsung device owners of GS2, GS3, OG Note, Note 2 & Note 10.1 beware of serious malware exploit. Retrieved March 31, 2013 from http://androidheadlines.com/2012/12/psa-samsung-device-owners-of-gs2-gs3-og-note-note-2-note-10-1-beware-of-serious-malware-exploit.html

Distefano, A., Me, Gianluigi., & Pace, F. (2010). Android anti-forensics through a local paradigm. Digital Investigation, 7, 83-94.

Ducrohet, X. (2010, December 6). Android 2.3 platform and updated SDK tools. Retrieved April 12, 2013 from http://android-developers.blogspot.com/2010/12/android-23-platform-and-updated-sdk.html

Ducrohet, X. (2011, October 18). Android 4.0 platform and updated SDK tools. Retrieved April 12, 2013 from http://android-developers.blogspot.com/2011/10/android-40-platform-and-updated-sdk.html

Eadicicco, L. (2012, December 4). Android will remain most popular mobile platform through 2016, IDC says. Retrieved 15 January, 2013 from http://www.digitaltrends.com/mobile/android-most-popular-platform-2016/#ixzz2I5uiYqs5

Elgin, B. (2005, August 16). Google buys Android for its mobile arsenal. Retrieved 11 April 2013 from http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal

F-Secure Labs. (2012, November 5). Mobile threat report Q3 2012. Retrieved 15 January, 2013 from http://www.f-secure.com/static/doc/labs_global/Research/Mobile%20Threat%20Report%20Q3%202012.pdf

Fingas, J. (2012, June 27). Google play hits 600,000 apps, 20 billion total installs. Retrieved January 15, 2013 from http://www.engadget.com/2012/06/27/ google-play-hits-600000-apps/

Ghosh, A. (2012, June 27). Introducing Android 4.1 (jelly bean) preview platform, and more. Retrieved April 12, 2013 from http://android-developers.blogspot.com/2012/06/introducing-android-41-jelly-bean.html

Google. (2011, November 3). Issue 21523: CVE-2011-1352: privilege escalation in PowerVR SGX drivers. Retrieved March 31, 2013 from https://code.google.com/p/android/issues/detail?id=21523

Google. (2012, November 29). Binaries for Nexus devices. Retrieved January 15, 2013 from https://developers.google.com/android/nexus/drivers

Greenberg, A. (2012, May 15). Antivirus firm: 75% of phone-based malware now targets Android. Retrieved July 15, 2012 from http://www.forbes.com/sites/andygreenberg/2012/05/15/antivirus-firm-75-of-phone-based-malware-now-targets-android/

HTC Corporation. (2008, September 23). T-Mobile unveils the T-Mobile G1 — the first phone powered by Android. Press Release. Retrieved December 1, 2009 from http://www.htc.com/www/press.aspx?id=66338&lang=1033

Hoog, A. (2009, May 29). Android Forensics: Android forensics [PowerPoint slides]. Retrieved December 1, 2009 from http://viaforensics.com.

Hoog, A. (2011). Android Forensics: Investigation, Analysis, and Mobile Security for Google Android. Waltham, Massachusetts: Syngress.

Imran, A. (2012, July 20). How to install / flash Android 4.1.1 jelly bean on Nexus S (i9020/i9020T/i9020A) [Tutorial]. Retrieved January 16, 2012 from http://www.redmondpie.com/how-to-install-flash-android-4.1.1-jelly-bean-on-nexus-s-i9020i9020ti9020a-tutorial/

IndefactorX. (8 December 2009). "Re: Droid rooted ! update.zip to root your droid." Android Community. http://androidcommunity.com/forums/f4/ droid-rooted-update-zip-to-root-your-droid-29398/

International Data Corporation. (2012, December 4). Worldwide mobile phone growth expected to drop to 1.4% in 2012 despite continued growth of smartphones, according to IDC. Press Release. Retrieved 15 January 2013 from http://www.idc.com/getdoc.jsp?containerId=prUS23818212#.UPX_Uc9QB2O

Kovacik, S., & O'Day, D.R. (2010). A proposed methodology for victim Android phone analysis by law enforcement investigators. Retrieved February 11, 2011 from http://digitollblog.com/Proposed%20Methodology%20for%20Android%20Forensics.pdf

Lessard, J., & Kessler, G.C. (2010). Android forensics: Simplifying cell phone examinations. Small Scale Digital Device Forensics Journal, 4(1).

Londatiga, L. (2010, July 2). How to create Android update zip package. Retrieved July 15, 2012 from http://www.londatiga.net/it/how-to-create-android-update-zip- package/

McGhee, R. (2010, January 22). Creating an Android update.zip package. Retrieved January 16, 2013 from http://www.robmcghee.com/android/creating-an-android-update-zip-package/

McKemmish, R. (1999). What is forensic computing? Trends & Issues in Crime and Criminal Justice, 118.

Micro Systemation AB. (2011, February 18). MSAB releases XRY 5.4. Retrieved 20 February 2011 from http://www.msab.com/support/news/archive/2011/february/page.php

Mohindra, D. (2008). “The Android Project”: Incident Response. Retrieved February 18, 2010 from http://www1.webng.com/dhruv/material/ android_report.pdf.

Mooij, B.: (2010, September 29). Data extraction from a physical dump. Retrieved February 1, 2012 from http://www.dfinews.com/article/data-extraction-physical-dump

Naraine, R. (2012, July 13). Android malware's dirty secret: repackaging of legit apps. Retrieved July 15, 2012 from http://www.zdnet.com/android-malwares-dirty-secret-repackaging-of-legit-apps-7000000886/

National Institute of Standards and Technology. (2007, May). Guidelines on cell phone forensics. U.S. Department of Commerce, National Institute of Standards and Technology. Retrieved 12 February, 2011 from http://csrc.nist.gov/publications/nistpubs/800-101/SP800-101.pdf

National Institute of Standards and Technology. (2010). Smart phone tool specification. Version 1.1. U.S. Department of Commerce, National Institute of Standards and Technology. Retrieved 12 September, 2010 from http://www.cftt.nist.gov/documents/Smart_Phone_Tool_Specification.pdf

Oxygen Software Company. (2011). Supported devices features matrix. Retrieved 20 February, 2011 from http://www.oxygen-forensic.com/en/models/

Paraben Corporation. (2010). Device Seizure supported models and plug-in details. Retrieved 20 February, 2011 from http://www.paraben.com/cell_models.html

Penguin.lin. (2012, August 15). Yaffs2utils. Google Code project. Retrieved February 1, 2013 from http://code.google.com/p/yaffs2utils/

Project on Scientific Knowledge and Public Policy. (2003). Daubert: The most influential Supreme Court ruling you've never heard of. Boston, MA.

Quick, D. & Alzaabi, M. (2001). Forensic analysis of the Android file system YAFFS2. Australian Digital Forensics Conference.

Robinson, S. (June 2003). Still guarding secrets, RSA earns accolades for its founders. SIAM News, 36(5).

Thompson, E. (2005). MD5 collisions and the impact on computer forensics. Digital Investigation, 2, 36 – 40.

Verizon. (n.d.). Hard Reset - Galaxy Nexus by Samsung. Retrieved January 16, 2013 from http://support.verizonwireless.com/clc/devices/knowledge_base.html?id=51446

ViaForensics. (February 1, 2011). ViaForensics' AFLogical tool is best for Android forensic investigations. Retrieved 21 February 2011 from http://viaforensics.com/viaforensics-articles/viaforensics-aflogical-tool-android-forensic-investigations.html

Weiss, T.R. (2012, July 14). Android cracks 50 percent as smartphone sales rise: Nielsen. Retrieved 15 July 2012 from http://www.eweek.com/c/a/Mobile-And-Wireless/Android-Cracks-50-Percent-as-Smartphone-Sales-Rise-Nielsen-862559/

Wilcox, J. (2011, February 9). Gartner: Android smartphone sales surged 888.8% in 2010. Retrieved 20 February, 2011 from http://www.betanews.com/joewilcox/article/Gartner-Android-smartphone-sales-surged-8888-in-2010/1297309933

Xda Developers. (2011, April 21). GingerBreak APK (root for GingerBread). Retrieved March 31, 2013 from http://forum.xda-developers.com/showthread.php?t=1044765

Xda Developers. (2011, October 10). Revolutionary - zergRush local root 2.2/2.3. Retrieved March 31, 2013 from http://forum.xda-developers.com/showthread.php?t=1296916

Xda Developers. (n.d.). ClockworkMod Recovery. Retrieved January 16, 2013 from http://forum.xda-developers.com/wiki/ClockworkMod_Recovery

Yang, H. (2012). Meta-inf files - digests, signature, and certificate. Retrieved January 16, 2013 from http://www.herongyang.com/Android/Project-META-INF-Files-Digest-Signature-and-Certificate.html

APPENDIX

APPENDIX

Code Snippets

The apps used in this study were written using the Android Software Development Kit for the 2.3 version of Android. Android apps are written in Java. Snippets of code in the source for these apps appears in the following sections.

Image Extraction Source Snippet

Each of the three devices had a slightly different device block directory. The method for creating a bitstream image was similar for each. The following snippet is essential to the image creation process. This snippet is generalized to work for any of the three devices. The path to the device block for the userdata partition for all three devices appears in this snippet. The update package rooted the device, so the "su" command was available.

```
// source: http://stackoverflow.com/questions/14566861/obtain-in-out-
streams-to-file-via-process-shell
  // for Nexus One:
  public static String USERDATA_LOC = "/dev/mtd/mtd5";
  // for Nexus S:
  public static String USERDATA_LOC = "/dev/block/platform/s3c-
sdhci.0/by-name/userdata";
  // For Xoom:
```

```
  public static String USERDATA_LOC = "/dev/block/platform/sdhci-
tegra.3/by-name/userdata";
  @Override
  public void onCreate(Bundle savedInstanceState) {
    try
    {
      String sdcard =
Environment.getExternalStorageDirectory().getAbsolutePath();
      Process p = Runtime.getRuntime().exec("su");
      DataOutputStream outs = new DataOutputStream(p.getOutputStream());
      String command = "cat " + USERDATA_LOC + " > " + sdcard +
"/userdata.img";
      outs.writeBytes(command + "\n");
      Toast.makeText(this,  "Image Successfully created",
Toast.LENGTH_LONG).show();
    }
    catch (Exception ex)
    {
      Log.d("exception in shell", ex.getMessage());
    }
```

Limited Extraction Source Snippet

The following Java source is critical to the app which attempted a limited extraction of data from the device. This app leveraged the Android API to extract web history from the device. The history is saved to a file on the device's external storage directory.

```
// source:  http://stackoverflow.com/questions/2577084/android-read-
browser-history
public class MainActivity extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    try
    {
      File sdcard = Environment.getExternalStorageDirectory();
      File outFile = new File(sdcard, "webHistory.csv");
      FileOutputStream fOut = new FileOutputStream(outFile);
      OutputStreamWriter myOutWriter = new OutputStreamWriter(fOut);
      Cursor histCursor = this.managedQuery(Browser.BOOKMARKS_URI,
Browser.HISTORY_PROJECTION, null, null, null);
      Log.d("Cursor count", String.valueOf(histCursor.getCount()));
      if (histCursor.moveToFirst() && histCursor.getCount() > 0)
      {
        myOutWriter.append("Title;URL;NumberOfVisits;LastVisitEpoch\n");
        while (histCursor.isAfterLast() == false)
        {

myOutWriter.append(histCursor.getString(Browser.HISTORY_PROJECTION_TITL
E_INDEX));
```

```
          myOutWriter.append(";");

myOutWriter.append(histCursor.getString(Browser.HISTORY_PROJECTION_URL_
INDEX));
          myOutWriter.append(";");
myOutWriter.append(String.valueOf(histCursor.getInt(Browser.HISTORY_PRO
JECTION_VISITS_INDEX)));
          myOutWriter.append(";");
myOutWriter.append(histCursor.getString(Browser.HISTORY_PROJECTION_DATE
_INDEX));
          myOutWriter.append("\n");
          histCursor.moveToNext();
        }
      }
      myOutWriter.close();
      fOut.close();
    }
    catch (Exception ex)
    {
      Log.d("Exception", ex.getMessage());
    }
  }
}
```

Malware Source Snippet

The following Java source is critical to the app which represented malware. This app attempted to download a large file from a local web server. The URL of the web server was redacted for publication.

```
// source: various pages on stackoverflow.com
public class MainActivity extends Activity {
  private static final String URLString =
"http://[redacted]/android/spaceWaster.txt"; // URL of web server,
redacted for publication
  private static final String dest = "spaceWaster.txt";
  private static final int DOWNLOAD_BUFFER_SIZE = 4096;
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    readWebpage();
  }
  private class DownloadWebPageTask extends AsyncTask<String, Void,
String>
  {
    @Override
    protected String doInBackground(String... urls)
    {
      String downloadedText = "";
      for (String url : urls)
      {
        DefaultHttpClient client = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(url);
```

```
        HttpResponse execute;
        try
        {
          execute = client.execute(httpGet);
          InputStream content;
          content = execute.getEntity().getContent();
          BufferedReader buffer = new BufferedReader(new
InputStreamReader(content));
          String s = "";
          while ((s = buffer.readLine()) != null)
          {
            downloadedText += s;
          }
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
      return downloadedText;
    }
  }
  public void readWebpage()
  {
    DownloadWebPageTask task = new DownloadWebPageTask();
    task.execute(new String[] {URLString});
  }
}
```

Update Package Scripts

The following sections are the contents of the file updater-script in each of the update packages. This file is located at /META-INF/com/google/android within each update package. In all of the packages, except for package 3, an app is installed. The .apk file, or the application file, is placed in the directory /data/app on the device. The .apk file is in the update package in the directory /data/app. The packages on the Xoom where an app was installed required an extra step where the app was granted proper permissions. The other two devices did not require this step.

Update Package 1 updater-script for Nexus One

```
ui_print("Mounting data...");
mount("yaffs2", "MTD", "userdata", "/data");
ui_print("Mounting system...");
mount("yaffs2", "MTD", "system", "/system");
ui_print("Data and system mounted, now copying...");
package_extract_dir("data", "/data");
package_extract_dir("system", "/system");
ui_print("Permissions ...");
set_perm(0, 2000, 06755, "/system/bin/su");
unmount("/data");
unmount("/system");
ui_print("Finished!");
```

Update Package 1 updater-script for Nexus S

```
ui_print("Mounting user data ....");
```

```
mount("ext4", "EMMC", "/dev/block/platform/s3c-sdhci.0/by-
name/userdata", "/data");
ui_print("Mounting system ....");
mount("ext4", "EMMC", "/dev/block/platform/s3c-sdhci.0/by-name/system",
"/system");
ui_print("Extracting data ....");
package_extract_dir("data", "/data");
ui_print("Extracting system ....");
package_extract_dir("system", "/system");
ui_print("Fixing permissions ....");
set_perm(0, 0, 06755, "/system/xbin/su");
ui_print("Unmounting user data ....");
unmount("/data");
ui_print("Unmounting system ....");
unmount("/system");
ui_print("All done!");
```

Update Package 1 updater-script for Xoom

```
ui_print("Mounting data...");
mount("ext4", "EMMC", "/dev/block/platform/sdhci-tegra.3/by-
name/userdata", "/data");
ui_print("Mounting system...");
mount("ext4", "EMMC", "/dev/block/platform/sdhci-tegra.3/by-
name/system", "/system");
ui_print("Data and system mounted, now copying...");
package_extract_dir("data", "/data");
package_extract_dir("system", "/system");
```

```
ui_print("Permissions ...");
set_perm(0, 2000, 06755, "/system/xbin/su");
set_perm(0, 2000, 0644, "/data/app/su.apk");
set_perm(0, 2000, 0644, "/data/app/imageUserdataXoom.apk");
ui_print("Unmounting ...");
unmount("/data");
unmount("/system");
ui_print("Finished!");
```

Update Package 2 updater-script for Nexus One

```
ui_print("Mounting data...");
mount("yaffs2", "MTD", "userdata", "/data");
ui_print("Data mounted, now copying...");
package_extract_dir("data", "/data");
unmount("/data");
ui_print("Finished!");
```

Update Package 2 updater-script for Nexus S

```
ui_print("Mounting data...");
mount("ext4", "EMMC", "/dev/block/platform/s3c-sdhci.0/by-
name/userdata", "/data");
ui_print("Data mounted, now copying...");
package_extract_dir("data", "/data");
unmount("/data");
ui_print("Finished!");
```

Update Package 2 updater-script for Xoom

```
ui_print("Mounting data...");
mount("ext4", "EMMC", "/dev/block/platform/sdhci-tegra.3/by-
name/userdata", "/data");
ui_print("Data mounted, now copying...");
package_extract_dir("data", "/data");
ui_print("Permissions ...");
set_perm(0, 2000, 0644, "/data/app/webHistory.apk");
unmount("/data");
ui_print("Finished!");
```

Update Package 3 updater-script for Nexus One

```
ui_print("Mounting data...");
mount("yaffs2", "MTD", "userdata", "/data");
ui_print("Now copying...");
package_extract_dir("data", "/data");
unmount("/data");
ui_print("Finished!");
```

Update Package 3 updater-script for Nexus S

```
ui_print("Mounting user data ....");
mount("ext4", "EMMC", "/dev/block/platform/s3c-sdhci.0/by-
name/userdata", "/data");
ui_print("Now copying ....");
package_extract_dir("data", "/data");
ui_print("Unmounting user data ....");
unmount("/data");
```

```
ui_print("All done!");
```

Update Package 3 updater-script for Xoom

```
ui_print("Mounting user data ....");
mount("ext4", "EMMC", "/dev/block/platform/sdhci-tegra.3/by-
name/userdata", "/data");
ui_print("Now copying ....");
package_extract_dir("data", "/data");
ui_print("Unmounting user data ....");
unmount("/data");
ui_print("All done!");
```

Update Package 4 updater-script for Nexus One

```
ui_print("Mounting data...");
mount("yaffs2", "MTD", "userdata", "/data");
ui_print("Data mounted, now copying...");
package_extract_dir("data", "/data");
unmount("/data");
ui_print("Finished!");
```

Update Package 4 updater-script for Nexus S

```
ui_print("Mounting data...");
mount("ext4", "EMMC", "/dev/block/platform/s3c-sdhci.0/by-
name/userdata", "/data");
ui_print("Data mounted, now copying...");
package_extract_dir("data", "/data");
```

```
unmount("/data");
ui_print("Finished!");
```

Update Package 4 updater-script for Xoom

```
ui_print("Mounting data...");
mount("ext4", "EMMC", "/dev/block/platform/sdhci-tegra.3/by-
name/userdata", "/data");
ui_print("Data mounted, now copying...");
package_extract_dir("data", "/data");
ui_print("Permissions ...");
set_perm(0, 2000, 0644, "/data/app/DownloadLargeFile.apk");
unmount("/data");
ui_print("Finished!");
```

Complete Data

A summary of the data collected in this experiment is presented in Chapter 4. The complete data set is presented in the following tables.

Table A.1 *Results for Update Package 1 on Nexus One, First Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Success	
A bitstream image of the userdata partition must be on the external storage media	Success	
The bistream image must be of the correct size for the device	Success	196.2 Megabytes
The bitstream image must contain a valid file system with no corruptions	Success	
Astro File Browser app and the Holy Bible app must be in the bitstream image	Success	
The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Success	
Overall determination	Success	

Table A.2 *Results for Update Package 1 on Nexus One, Second Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Success	
A bitstream image of the userdata partition must be on the external storage media	Success	
The bistream image must be of the correct size for the device	Success	196.2 Megabytes
The bitstream image must contain a valid file system with no corruptions	Success	
Astro File Browser app and the Holy Bible app must be in the bitstream image	Success	
The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Success	
Overall determination	Success	

Table A.3 *Results for Update Package 1 on Nexus One, Third Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Success	
A bitstream image of the userdata partition must be on the external storage media	Success	
The bistream image must be of the correct size for the device	Success	196.2 Megabytes
The bitstream image must contain a valid file system with no corruptions	Success	
Astro File Browser app and the Holy Bible app must be in the bitstream image	Success	
The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Success	
Overall determination	Success	

Table A.4 *Results for Update Package 1 on Nexus One, Fourth Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Success	
A bitstream image of the userdata partition must be on the external storage media	Success	
The bistream image must be of the correct size for the device	Success	196.2 Megabytes
The bitstream image must contain a valid file system with no corruptions	Success	
Astro File Browser app and the Holy Bible app must be in the bitstream image	Success	
The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Success	
Overall determination	Success	

Table A.5 *Results for Update Package 1 on Nexus One, Fifth Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Success	
A bitstream image of the userdata partition must be on the external storage media	Success	
The bistream image must be of the correct size for the device	Success	196.2 Megabytes
The bitstream image must contain a valid file system with no corruptions	Success	
Astro File Browser app and the Holy Bible app must be in the bitstream image	Success	
The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Success	
Overall determination	Success	

Table A.6 *Results for Update Package 1 on Nexus S, First Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Fail	Imaging application did not start automatically. Upon manually opening the application and then restarting the device the application opened automatically and successfully created the image.
A bitstream image of the userdata partition must be on the external storage media	Success	
The bistream image must be of the correct size for the device	Success	1 Gigabyte
The bitstream image must contain a valid file system with no corruptions	Success	
Astro File Browser app and the Holy Bible app must be in the bitstream image	Success	

Table A.6 Continued

The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Success
Overall determination	Fail

Table A.7 *Results for Update Package 1 on Nexus S, Second Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Fail	Imaging application did not start automatically. Upon manually opening the application and then restarting the device the application opened automatically and successfully created the image.
A bitstream image of the userdata partition must be on the external storage media	Success	
The bistream image must be of the correct size for the device	Success	1 Gigabyte
The bitstream image must contain a valid file system with no corruptions	Success	
Astro File Browser app and the Holy Bible app must be in the bitstream image	Success	

Table A.7 Continued

The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Success
Overall determination	Fail

Table A.8 *Results for Update Package 1 on Nexus S, Third Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Fail	Imaging application did not start automatically. Upon manually opening the application and then restarting the device the application opened automatically and successfully created the image.
A bitstream image of the userdata partition must be on the external storage media	Success	
The bistream image must be of the correct size for the device	Success	1 Gigabyte
The bitstream image must contain a valid file system with no corruptions	Success	
Astro File Browser app and the Holy Bible app must be in the bitstream image	Success	

Table A.8 Continued

The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Success
Overall determination	Fail

Table A.9 *Results for Update Package 1 on Nexus S, Fourth Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Fail	Imaging application did not start automatically. Upon manually opening the application and then restarting the device the application opened automatically and successfully created the image.
A bitstream image of the userdata partition must be on the external storage media	Success	
The bistream image must be of the correct size for the device	Success	1 Gigabyte
The bitstream image must contain a valid file system with no corruptions	Success	
Astro File Browser app and the Holy Bible app must be in the bitstream image	Success	

Table A.9 Continued

The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Success
Overall determination	Fail

Table A.10 *Results for Update Package 1 on Nexus S, Fifth Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Fail	Imaging application did not start automatically. Upon manually opening the application and then restarting the device the application opened automatically and successfully created the image.
A bitstream image of the userdata partition must be on the external storage media	Success	
The bistream image must be of the correct size for the device	Success	1 Gigabyte
The bitstream image must contain a valid file system with no corruptions	Success	
Astro File Browser app and the Holy Bible app must be in the bitstream image	Success	

Table A.10 Continued

The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Success
Overall determination	Fail

Table A.11 *Results for Update Package 1 on Xoom, First Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Fail	Imaging application did not start automatically. Upon manually opening the application and then restarting the device the application opened automatically and successfully created the image
A bitstream image of the userdata partition must be on the external storage media	Fail	Device could not detect SD Card; Could not create an image
The bistream image must be of the correct size for the device	Fail	The image was not properly created
The bitstream image must contain a valid file system with no corruptions	Fail	The image was not properly created
Astro File Browser app and the Holy Bible app must be in the bitstream image	Fail	The image was not properly created

Table A.11 Continued

The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Fail	The image was not properly created
Overall determination	Fail	

Table A.12 *Results for Update Package 1 on Xoom, Second Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Fail	Imaging application did not start automatically. Upon manually opening the application and then restarting the device the application opened automatically and successfully created the image.
A bitstream image of the userdata partition must be on the external storage media	Fail	Device could not detect SD Card; Could not create an image
The bistream image must be of the correct size for the device	Fail	The image was not properly created
The bitstream image must contain a valid file system with no corruptions	Fail	The image was not properly created
Astro File Browser app and the Holy Bible app must be in the bitstream image	Fail	The image was not properly created

Table A.12 Continued

The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Fail	The image was not properly created
Overall determination	Fail	

Table A.13 *Results for Update Package 1 on Xoom, Third Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Fail	Imaging application did not start automatically. Upon manually opening the application and then restarting the device the application opened automatically and successfully created the image.
A bitstream image of the userdata partition must be on the external storage media	Fail	Device could not detect SD Card; Could not create an image
The bistream image must be of the correct size for the device	Fail	The image was not properly created
The bitstream image must contain a valid file system with no corruptions	Fail	The image was not properly created
Astro File Browser app and the Holy Bible app must be in the bitstream image	Fail	The image was not properly created

Table A.13 Continued

The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Fail	The image was not properly created
Overall determination	Fail	

Table A.14 *Results for Update Package 1 on Xoom, Fourth Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Fail	Imaging application did not start automatically. Upon manually opening the application and then restarting the device the application opened automatically and successfully created the image.
A bitstream image of the userdata partition must be on the external storage media	Fail	Device could not detect SD Card; Could not create an image
The bistream image must be of the correct size for the device	Fail	The image was not properly created
The bitstream image must contain a valid file system with no corruptions	Fail	The image was not properly created
Astro File Browser app and the Holy Bible app must be in the bitstream image	Fail	The image was not properly created

Table A.14 Continued

The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Fail	The image was not properly created
Overall determination	Fail	

Table A.15 *Results for Update Package 1 on Xoom, Fifth Execution*

	Results (Success or Fail)	Notes
The application creating the bitstream image must open upon and begin imaging upon boot	Fail	Imaging application did not start automatically. Upon manually opening the application and then restarting the device the application opened automatically and successfully created the image.
A bitstream image of the userdata partition must be on the external storage media	Fail	Device could not detect SD Card; Could not create an image
The bistream image must be of the correct size for the device	Fail	The image was not properly created
The bitstream image must contain a valid file system with no corruptions	Fail	The image was not properly created
Astro File Browser app and the Holy Bible app must be in the bitstream image	Fail	The image was not properly created

Table A.15 Continued

The web browsing history must contain the websites www.purdue.edu, developer.android.com, and www.nfl.com	Fail	The image was not properly created
Overall determination	Fail	

Table A.16 *Results for Update Package 2 on Nexus One, First Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	Displayed multiple entries for each of the four websites; perhaps because two were redirected to mobile versions of the site, and mistyped some if not all of them; pages designated as "favorites" by Android also were in here
Overall determination	Success	

Table A.17 *Results for Update Package 2 on Nexus One, Second Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	Displayed multiple entries for only two; these two redirected to mobile version of the site. No mistypes. Again had favorites.
Overall determination	Success	

Table A.18 *Results for Update Package 2 on Nexus One, Third Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	Displayed multiple entries for only two; these two redirected to mobile version of the site. No mistypes. Again had favorites.
Overall determination	Success	

Table A.19 *Results for Update Package 2 on Nexus One, Fourth Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	Some extraneous entries due to mistypes.
Overall determination	Success	

Table A.20 *Results for Update Package 2 on Nexus One, Fifth Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	Displayed multiple entries for only two; these two redirected to mobile version of the site. No mistypes. Again had favorites.
Overall determination	Success	

Table A.21 *Results for Update Package 2 on Nexus S, First Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	Displayed multiple entries for some of the websites, perhaps because of redirection to mobile site. Contained favorites as designated by Android.
Overall determination	Success	

Table A.22 *Results for Update Package 2 on Nexus S, Second Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	Displayed multiple entries for some of the websites, perhaps because of redirection to mobile site. Contained favorites as designated by Android.
Overall determination	Success	

Table A.23 *Results for Update Package 2 on Nexus S, Third Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	Displayed multiple entries for some of the websites, perhaps because of redirection to mobile site. Contained favorites as designated by Android.
Overall determination	Success	

Table A.24 *Results for Update Package 2 on Nexus S, Fourth Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	Displayed multiple entries for some of the websites, perhaps because of redirection to mobile site. Contained favorites as designated by Android.
Overall determination	Success	

Table A.25 *Results for Update Package 2 on Nexus S, Fifth Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	Displayed multiple entries for some of the websites, perhaps because of redirection to mobile site. Contained favorites as designated by Android.
Overall determination	Success	

Table A.26 *Results for Update Package 2 on Xoom, First Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	No repeat websites. Contained favorites as designated by Android.
Overall determination	Success	

Table A.27 *Results for Update Package 2 on Xoom, Second Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	No repeat websites. Contained favorites as designated by Android.
Overall determination	Success	

Table A.28 *Results for Update Package 2 on Xoom, Third Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	No repeat websites. Contained favorites as designated by Android.
Overall determination	Success	

Table A.29 *Results for Update Package 2 on Xoom, Fourth Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	No repeat websites. Contained favorites as designated by Android.
Overall determination	Success	

Table A.30 *Results for Update Package 2 on Xoom, Fifth Execution*

	Results (Success or Fail)	Notes
A CSV file must be on the device's external storage media containing the web browsing history	Success	
The web browsing history must include the websites www.att.com, www.sprint.com, www.verizon.com, and www.t-mobile.com	Success	No repeat websites. Contained favorites as designated by Android.
Overall determination	Success	

Table A.31 *Results for Update Package 3 on Nexus One, First Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Success	
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Success	
The browser must open and execute properly	Success	
Overall determination	Success	

Table A.32 *Results for Update Package 3 on Nexus One, Second Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Success	
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Success	
The browser must open and execute properly	Success	
Overall determination	Success	

Table A.33 *Results for Update Package 3 on Nexus One, Third Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Success	
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Success	
The browser must open and execute properly	Success	
Overall determination	Success	

Table A.34 *Results for Update Package 3 on Nexus One, Fourth Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Success	
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Success	
The browser must open and execute properly	Success	
Overall determination	Success	

Table A.35 *Results for Update Package 3 on Nexus One, Fifth Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Success	
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Success	
The browser must open and execute properly	Success	
Overall determination	Success	

Table A.36 *Results for Update Package 3 on Nexus S, First Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Fail	Browser history was cleared
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Fail	Browser history was cleared
The browser must open and execute properly	Fail	Browser crashed upon opening
Overall determination	Fail	

Table A.37 *Results for Update Package 3 on Nexus S, Second Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Fail	Browser history was cleared
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Fail	Browser history was cleared
The browser must open and execute properly	Fail	Browser crashed upon opening
Overall determination	Fail	

Table A.38 *Results for Update Package 3 on Nexus S, Third Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Fail	Browser history was cleared
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Fail	Browser history was cleared
The browser must open and execute properly	Fail	Browser crashed upon opening
Overall determination	Fail	

Table A.39 *Results for Update Package 3 on Nexus S, Fourth Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Fail	Browser history was cleared
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Fail	Browser history was cleared
The browser must open and execute properly	Fail	Browser crashed upon opening
Overall determination	Fail	

Table A.40 *Results for Update Package 3 on Nexus S, Fifth Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Fail	Browser history was cleared
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Fail	Browser history was cleared
The browser must open and execute properly	Fail	Browser crashed upon opening
Overall determination	Fail	

Table A.41 *Results for Update Package 3 on Xoom, First Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Success	
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Fail	Web history remained unchanged
The browser must open and execute properly	Success	
Overall determination	Fail	

Table A.42 *Results for Update Package 3 on Xoom, Second Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Success	
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Fail	Web history remained unchanged
The browser must open and execute properly	Success	
Overall determination	Fail	

Table A.43 *Results for Update Package 3 on Xoom, Third Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Success	
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Fail	Web history remained unchanged
The browser must open and execute properly	Success	
Overall determination	Fail	

Table A.44 *Results for Update Package 3 on Xoom, Fourth Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Success	
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Fail	Web history remained unchanged
The browser must open and execute properly	Success	
Overall determination	Fail	

Table A.45 *Results for Update Package 3 on Xoom, Fifth Execution*

	Results (Success or Fail)	Notes
The device's browser history must include the pages www.purdue.edu, developer.android.com, and www.nfl.com, as entered for the first update package	Success	
The device's browser history must also include the websites www.bengals.com, www.microsoft.com, and www.walmart.com	Fail	Web history remained unchanged
The browser must open and execute properly	Success	
Overall determination	Fail	

Table A.46 *Results for Update Package 4 on Nexus One, First Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Success	
"spaceWaster.txt" must be 32 megabytes	Success	
The MD5 hash of "spaceWaster.txt" must match the original	Success	Hash of downloaded file: 279e4d151bcf4cbd9e2aab738b230aa7
Overall determination	Success	

Table A.47 *Results for Update Package 4 on Nexus One, Second Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Success	
"spaceWaster.txt" must be 32 megabytes	Success	
The MD5 hash of "spaceWaster.txt" must match the original	Success	Hash of downloaded file: 279e4d151bcf4cbd9e2aab738b230aa7
Overall determination	Success	

Table A.48 *Results for Update Package 4 on Nexus One, Third Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Success	
"spaceWaster.txt" must be 32 megabytes	Success	
The MD5 hash of "spaceWaster.txt" must match the original	Success	Hash of downloaded file: 279e4d151bcf4cbd9e2aab738b230aa7
Overall determination	Success	

Table A.49 *Results for Update Package 4 on Nexus One, Fourth Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Success	
"spaceWaster.txt" must be 32 megabytes	Success	
The MD5 hash of "spaceWaster.txt" must match the original	Success	Hash of downloaded file: 279e4d151bcf4cbd9e2aab738b230aa7
Overall determination	Success	

Table A.50 *Results for Update Package 4 on Nexus One, Fifth Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Success	
"spaceWaster.txt" must be 32 megabytes	Success	
The MD5 hash of "spaceWaster.txt" must match the original	Success	Hash of downloaded file: 279e4d151bcf4cbd9e2aab738b230aa7
Overall determination	Success	

Table A.51 *Results for Update Package 4 on Nexus S, First Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Fail	App crashed upon opening and did not download the file
"spaceWaster.txt" must be 32 megabytes	Fail	
The MD5 hash of "spaceWaster.txt" must match the original	Fail	
Overall determination	Fail	

Table A.52 *Results for Update Package 4 on Nexus S, Second Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Fail	App crashed upon opening and did not download the file
"spaceWaster.txt" must be 32 megabytes	Fail	
The MD5 hash of "spaceWaster.txt" must match the original	Fail	
Overall determination	Fail	

Table A.53 *Results for Update Package 4 on Nexus S, Third Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Fail	App crashed upon opening and did not download the file
"spaceWaster.txt" must be 32 megabytes	Fail	
The MD5 hash of "spaceWaster.txt" must match the original	Fail	
Overall determination	Fail	

Table A.54 *Results for Update Package 4 on Nexus S, Fourth Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Fail	App crashed upon opening and did not download the file
"spaceWaster.txt" must be 32 megabytes	Fail	
The MD5 hash of "spaceWaster.txt" must match the original	Fail	
Overall determination	Fail	

Table A.55 *Results for Update Package 4 on Nexus S, Fifth Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Fail	App crashed upon opening and did not download the file
"spaceWaster.txt" must be 32 megabytes	Fail	
The MD5 hash of "spaceWaster.txt" must match the original	Fail	
Overall determination	Fail	

Table A.56 *Results for Update Package 4 on Xoom, First Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Fail	App crashed upon opening and did not download the file
"spaceWaster.txt" must be 32 megabytes	Fail	
The MD5 hash of "spaceWaster.txt" must match the original	Fail	
Overall determination	Fail	

Table A.57 *Results for Update Package 4 on Xoom, Second Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Fail	App crashed upon opening and did not download the file
"spaceWaster.txt" must be 32 megabytes	Fail	
The MD5 hash of "spaceWaster.txt" must match the original	Fail	
Overall determination	Fail	

Table A.58 *Results for Update Package 4 on Xoom, Third Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Fail	App crashed upon opening and did not download the file
"spaceWaster.txt" must be 32 megabytes	Fail	
The MD5 hash of "spaceWaster.txt" must match the original	Fail	
Overall determination	Fail	

Table A.59 *Results for Update Package 4 on Xoom, Fourth Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Fail	App crashed upon opening and did not download the file
"spaceWaster.txt" must be 32 megabytes	Fail	
The MD5 hash of "spaceWaster.txt" must match the original	Fail	
Overall determination	Fail	

Table A.60 *Results for Update Package 4 on Xoom, Fifth Execution*

	Results (Success or Fail)	Notes
The file "spaceWaster.txt" must be on the device's external storage	Fail	App crashed upon opening and did not download the file
"spaceWaster.txt" must be 32 megabytes	Fail	
The MD5 hash of "spaceWaster.txt" must match the original	Fail	
Overall determination	Fail	